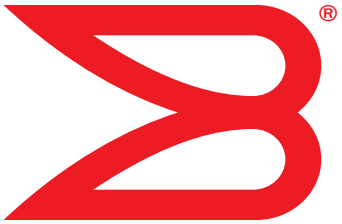
53-1002444-02
June 2012



ServerIron ADX

NAT64 Configuration Guide

Supporting Brocade ServerIron ADX version 12.4.00

BROCADE

©© 2012 Brocade Communications Systems, Inc. All Rights Reserved.

Brocade, Brocade Assurance, the B-wing symbol, DCX, Fabric OS, MLX, SAN Health, VCS, and VDX are registered trademarks, and AnyIO, Brocade One, CloudPlex, Effortless Networking, ICX, NET Health, OpenScript, and The Effortless Network are trademarks of Brocade Communications Systems, Inc., in the United States and/or in other countries. Other brands, products, or service names mentioned may be trademarks of their respective owners.

Notice: This document is for informational purposes only and does not set forth any warranty, expressed or implied, concerning any equipment, equipment feature, or service offered or to be offered by Brocade. Brocade reserves the right to make changes to this document at any time, without notice, and assumes no responsibility for its use. This informational document describes features that may not be currently available. Contact a Brocade sales office for information on feature and product availability. Export of technical data contained in this document may require an export license from the United States government.

The authors and Brocade Communications Systems, Inc. shall have no liability or responsibility to any person or entity with respect to any loss, cost, liability, or damages arising from the information contained in this book or the computer programs that accompany it.

The product described by this document may contain "open source" software covered by the GNU General Public License or other open source license agreements. To find out which open source software is included in Brocade products, view the licensing terms applicable to the open source software, and obtain a copy of the programming source code, please visit

http://www.brocade.com/support/oscd.

Brocade Communications Systems, Incorporated

Corporate and Latin American Headquarters
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134
Tel: 1-408-333-8000
Fax: 1-408-333-8101
E-mail: info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems China HK, Ltd.
No. 1 Guanghua Road
Chao Yang District
Units 2718 and 2818
Beijing 100020, China
Tel: +8610 6588 8888
Fax: +8610 6588 9999
E-mail: china-info@brocade.com

European Headquarters
Brocade Communications Switzerland Sàrl
Centre Swissair
Tour B - 4ème étage
29, Route de l'Aéroport
Case Postale 105
CH-1215 Genève 15
Switzerland
Tel: +41 22 799 5640
Fax: +41 22 799 5641
E-mail: emea-info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems Co., Ltd. (Shenzhen WFOE)
Citic Plaza
No. 233 Tian He Road North
Unit 1308 – 13th Floor
Guangzhou, China
Tel: +8620 3891 2000
Fax: +8620 3891 2111
E-mail: china-info@brocade.com

Document History

Title	Publication number	Summary of changes	Date
ServerIron ADX NAT64 Configuration Guide	*53-1002444-01*	New document	January 2012
ServerIron ADX NAT64 Configuration Guide	*53-1002444-02*	Document updated to correct errors and make improvement based on customer input.	June 2012

Contents

About This Document

Audience ix
Supported hardware and software ix
Document conventions ix
Text formatting ix
Command syntax conventions x
Notes, cautions, and danger notices x
Notice to the reader xi
Related publications xi
Getting technical help xi

Chapter 1 NAT64 and NAT46 Overview

In this chapter 1
Overview of NAT64 and NAT46 1
Stateful NAT64 translation 1
Stateless NAT64 translation 1
Stateless NAT46 translation 2
NAT64 and NAT46 implementation details 2
Requirements for all NAT64 and NAT46 configurations 2
Requirements for stateful and stateless NAT64 configurations 2
Requirements for stateless NAT46 configurations 3
Protocol support 3
NAT64 ICMPv6 to ICMPv4 message handling 3
NAT64 full-sized packet handling 5
NAT64 fragmentation support 5
Installing NAT64 on a previously configured ServerIron ADX . . . 6
References 6

Chapter 2 Stateful NAT64 Configuration

In this chapter 7
Stateful NAT64 overview 7
Stateful NAT64 components 7
Operation of stateful NAT64 8
Basic stateful NAT64 configuration 9
Configuring stateful IPv6 prefixes 9
Configuring basic IPv4 NAT address pools 10

Advanced stateful NAT64 configuration 11
Configuring stateful NAT64 with route injection 11
NAT64 sticky session configuration 15
Enabling connection logging 16
Configuring HTTP client IP address insertion 16
Configuring NAT64 packet fragmentation options 17
High availability for stateful NAT64 18
Configuring the port pool range for HA 18
Enabling inject active only for HA 19
Displaying NAT64 information 19
Displaying session information 19
Displaying NAT64 translations 20
Displaying NAT64 statistics 21
Displaying NAT64 resources 25
Clearing stateful NAT64 information 26
Debugging stateful NAT64 configurations 26

Chapter 3 Stateless NAT64 Configuration

In this chapter 27
Stateless NAT64 overview 27
Stateless NAT64 components 27
Operation of stateless NAT64 28
Populating the NAT64 mapping table 30
Stateless NAT64 static mapping configuration 30
Basic stateless NAT64 static mapping configuration 31
Stateless NAT64 static mapping with route injection 32
Stateless NAT64 packet fragmentation configuration 35
Stateless NAT64 dynamic mapping configuration 36
NAT64 real-time dynamic mapping configuration 36
NAT64 prefetched dynamic mapping configuration 38
High availability for NAT64 40
Displaying NAT64 information 40
Displaying NAT64 IPv6 to IPv4 address mappings 40
Displaying in-progress dynamic NAT64 mappings 41
Clearing NAT64 information 41
Clearing dynamic IPv4-IPv6 mappings 41
Clearing stateless NAT64 statistics 42
Debugging stateless NAT64 configurations 42

Chapter 4 Stateless NAT46 Configuration

In this chapter 43
Stateless NAT46 overview 43
Stateless NAT46 components 43
Operation of stateless NAT46 44
Populating the NAT46 mapping table 45

NAT46 static mapping configuration 45
Basic NAT46 static mapping configuration 46
Stateless NAT46 static mapping with route injection 47
Configuring NAT46 packet fragmentation 51
Stateless NAT46 dynamic mapping configuration 52
Real-time NAT46 dynamic mapping configuration 52
Prefetched NAT46 dynamic mapping configuration 54
High availability for NAT46 55
Displaying NAT46 information 56
Displaying NAT46 IPv4 to IPv6 address mappings 56
Displaying in-progress dynamic NAT46 mappings. 57
Clearing NAT46 information 57
Clearing dynamic IPv6-IPv4 mappings 57
Clearing stateless NAT46 statistics 58
Debugging NAT46 configurations 58

Chapter 5 Access Control Lists

In this chapter 59
How ServerIron ADX ADX processes ACLs 59
Prior to release 12.3.01 59
Beginning with release 12.3.01 and later 60
Rule-based ACLs 60
Processing of fragmented packets 61
Default ACL action 62
ACL IDs and entries 62
ACL entries and the Layer 4 CAM. 63
Aging out of entries in the Layer 4 CAM 63
Displaying the number of Layer 4 CAM entries 64
Specifying the maximum number of CAM entries 64
Configuring rule-based ACLs 65
Configuring standard numbered ACLs 65
Configuring extended numbered ACLs 67
Configuring standard or extended named ACLs 72
Modifying rule-based ACLs 73
Reordering ACLs. 74
Applying ACLs to interfaces 75
Adding, replacing, or deleting comments to rule-based ACLs 76
Adding comments to numbered ACLs 76
Replacing comments applied to numbered ACLs 76
Deleting comments applied to numbered ACLs 77
Adding comments to named ACLs 77
Replacing comments applied to named ACLs 78
Deleting comments applied to named ACLs 78

Displaying rule-based ACL entries 78
Displaying ACLs using the show access-list command 78
Displaying ACLs using the show ip access-lists command 79
Displaying ACLs using keywords 79
Displaying ACL bindings. 82
ACL logging 82
Syslog message for changed ACL mode. 83
Copying denied traffic to a mirror port for monitoring. 83
Displaying ACL log entries 83
Displaying ACL statistics for flow-based ACLs 84
Clearing flow-based ACL statistics 85
Dropping all fragments that exactly match a flow-based ACL 85
Clearing the ACL statistics. 85
Enabling ACL filtering of fragmented packets. 86
Filtering fragmented packets for rule-based ACLs. 86
Throttling the fragment rate 86
Enabling filtering for packets denied by flow-based ACLs 88
Enabling strict TCP or UDP mode for flow-based ACLs. 88
Enabling strict TCP mode 89
Enabling strict UDP mode 90
Configuring ACL packet and flow counters. 90
ACLs and ICMP 91
Using flow-based ACLs to filter ICMP packets 91
ICMP filtering with flow-based ACLs 92
Using flow-based ACLs and NAT on the same interface. 95
Troubleshooting rule-based ACLs. 96

Chapter 6 IPv6 Access Control Lists

In this chapter 97
IPv6 ACL overview 97
Configuration notes 98
Processing of IPv6 ACLs 98
Configuring IPv6 ACLs. 99
Example configurations. 99
Default and implicit IPv6 ACL actions. 101
IPv6 ACL syntax 102
Applying IPv6 ACLs to interfaces 105
Displaying IPv6 ACLs. 106
Displaying IPv6 ACLs bound to an interface 106
Using an ACL to restrict SSH access 107
Using an ACL to restrict Telnet access. 107
Logging IPv6 ACLs. 107

Chapter 7 Network Address Translation

In this chapter 109

Configuring NAT 110
- PAT 110
- Configuring static NAT 111
- Configuring dynamic NAT 111
- Enabling IP NAT 112
- NAT configuration examples 113

Forwarding packets without NAT translation 117

IP NAT with VIP overlap 117

Translation timeouts 118
- Configuring the NAT translation aging timer 119

Disabling IP NAT sticky behavior 119

Deleting IP NAT sticky sessions 120

Stateless static IP NAT 120

IP NAT redundancy 121
- Configuring static NAT entry in a HA setup 121
- Configuring dynamic NAT pool in a HA setup 121
- IP NAT session synchronization in HA configurations 132

Displaying NAT information 132
- Displaying NAT statistics 132
- Displaying NAT translation 135
- Displaying NAT redundancy information 136
- Displaying VRRPE information 136

Clearing NAT entries from the table 137

About This Document

Audience

This document is designed for system administrators with a working knowledge of Layer 2 and Layer 3 switching and routing.

If you are using a Brocade Layer 3 Switch, you should be familiar with the following protocols if applicable to your network: IP, RIP, OSPF, BGP, ISIS, IGMP, PIM, DVMRP, and VRRP.

Supported hardware and software

Although many different software and hardware configurations are tested and supported by Brocade Communications Systems, Inc. for 12.4.00 documenting all possible configurations and scenarios is beyond the scope of this document.

The following hardware platforms are supported by this release of this guide:

- ServerIron ADX 1000
- ServerIron ADX 4000
- ServerIron ADX 8000
- ServerIron ADX 10000

NOTE
The NAT64 gateway cannot be combined to other ServerIron ADX features such as SLB, GSLB, TCS, FWLB etc. This guide includes all features that can be enabled with a NAT64 gateway.

Document conventions

This section describes text formatting conventions and important notice formats used in this document.

Text formatting

The narrative-text formatting conventions that are used are as follows:

bold text	Identifies command names Identifies the names of user-manipulated GUI elements Identifies keywords Identifies text to enter at the GUI or CLI
italic text	Provides emphasis Identifies variables Identifies document titles
code text	Identifies CLI output

For readability, command names in the narrative portions of this guide are presented in bold: for example, **show version**.

Command syntax conventions

Command syntax in this manual follows these conventions:

command and **parameters**	Commands and parameters are printed in bold.
[ ]	Optional parameter.
variable	Variables are printed in italics enclosed in angled brackets < >.
...	Repeat the previous element, for example “member[;member...]”
|	Choose one of the parameters.

Notes, cautions, and danger notices

The following notices and statements are used in this manual. They are listed below in order of increasing severity of potential hazards.

NOTE
A note provides a tip, guidance or advice, emphasizes important information, or provides a reference to related information.

CAUTION

A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

DANGER

A Danger statement indicates conditions or situations that can be potentially lethal or extremely hazardous to you. Safety labels are also attached directly to products to warn of these conditions or situations.

Notice to the reader

This document may contain references to the trademarks of the following corporations. These trademarks are the properties of their respective companies and corporations.

These references are made for informational purposes only.

Corporation	Referenced Trademarks and Products
Sun Microsystems	Solaris
Microsoft Corporation	Windows NT, Windows 2000
The Open Group	Linux

Related publications

The following Brocade documents supplement the information in this guide:

- *Release Notes for ServerIron Switch and Router Software TrafficWorks 12.4.00*
- *ServerIron ADX Graphical User Interface*
- *ServerIron ADX Server Load Balancing Guide*
- *ServerIron ADX Advanced Server Load Balancing Guide*
- *ServerIron ADX Global Server Load Balancing Guide*
- *ServerIron ADX Security Guide*
- *ServerIron ADX Administration Guide*
- *ServerIron ADX Switch and Router Guide*
- *ServerIron ADX Firewall Load Balancing Guide*
- *ServerIron ADX Hardware Installation Guide*
- *IronWare MIB Reference*

Getting technical help

To contact Technical Support, go to http://www.brocade.com/services-support/index.page for the latest e-mail and telephone contact information.

Chapter

1 NAT64 and NAT46 Overview

In this chapter

- Overview of NAT64 and NAT46 1
- NAT64 and NAT46 implementation details 2

Overview of NAT64 and NAT46

The industry faces a challenging transition while it moves from its current IPv4-capable routers, switches, servers, and applications to IPv6-ready devices. Service providers—whether they are providing content, hosting services, or Internet access—cannot add or accommodate new customers unless their content is equally accessible to both IPv4 and IPv6 users.

The ServerIron ADX supports both current IPv4 customers and offers a seamless experience to users that access IPv4 Internet services through new IPv6-only networks in three simple and cost effective ways: stateful NAT64 gateway, stateless NAT64 gateway, and a stateless NAT46 gateway.

Stateful NAT64 translation

NAT64 capabilities of Brocade ServerIron ADX enable organizations to bring new IPv6 customers onboard while utilizing their existing IPv4-based infrastructures.

In a stateful NAT64 configuration, a ServerIron ADX configured as a NAT64 gateway enables IPv6-only clients to communicate with IPv4 resources by way of address and packet translation. This translation operation is performed on the NAT64 gateway using stateful sessions.

NAT64 is transparent to end users, meaning that IPv6-only clients can access IPv4 resources seamlessly. NAT64 translation does not require any change on CPE devices at the client end.

NAT64 requires the IPv6 client address be replaced by an IPv4 address while communicating with the IPv4 network and resources. IPv4 addresses are configured on the ServerIron ADX. Stateful NAT64 maps multiple IPv6 clients to each of these IPv4 addresses using a technique similar to NAT-PT. The translation operation requires knowledge of the IPv6-IPv4 address and port mapping and this is maintained using stateful sessions.

Stateless NAT64 translation

Stateless NAT64 enables IPv6-only clients to communicate with IPv4-only hosts similar to stateful NAT64. However, unlike stateful NAT64, it requires a one-to-one mapping between the IPv6 and the IPv4 address of clients. In effect, stateless NAT64 does not require the creation of flow sessions because an IPv4 address is allocated for each IPv6 address and there is no dynamic mapping of IPv4 addresses to IPv6 addresses.

The stateless NAT64 gateway generates an IPv4 source address for the IPv6 host from the IPv4-IPv6 mapping table. The mapping table can be dynamically constructed using a DNS server or manually configured. The IPv4 destination address is obtained by stripping away the IPv6 prefix from the synthesized IPv6 destination address provided by the DNS64 server.

Stateless NAT46 translation

When configured as a stateless NAT46 gateway, the ServerIron ADX enables IPv4-only clients to communicate with IPv6-only resources using address and packet translation. In a stateless NAT46 configuration, the ServerIron ADX uses a mapping table to match IPv4 request packets sent from the IPv4 clients with the IPv6 destination address of the IPv6 resource. The mapping table can be dynamically constructed using a DNS server or manually configured.

NAT46 translation is offered in stateless mode only.

NAT64 and NAT46 implementation details

This section describes the components required for a NAT64 or NAT46 implementation and provides information that is common to the stateful NAT64 configuration described in Chapter 2, “Stateful NAT64 Configuration”, the stateless NAT64 configuration described in Chapter 3, “Stateless NAT64 Configuration”, and the stateless NAT46 configuration described in Chapter 4, “Stateless NAT46 Configuration”.

NOTE
A ServerIron ADX configured as either a NAT64 gateway or a NAT46 gateway cannot be configured for other ServerIron ADX features such as SLB, GSLB, TCS, FWLB, and so on. This guide describes all features that can be enabled on the NAT64 and NAT46 gateway.

Requirements for all NAT64 and NAT46 configurations

The NAT64 mechanism is implemented on a ServerIron ADX that has (at least) two interfaces: an IPv4 interface connected to the IPv4 network, and an IPv6 interface connected to the IPv6 network. IPv6 must be enabled on the ServerIron ADX.

Requirements for stateful and stateless NAT64 configurations

The following components are required for stateful and stateless NAT64 gateway configurations:

- The DNS64 resolver provides AAAA responses for any resource that has an IPv4 address but not an IPv6 address. It generates the IPv6 address for the resource by concatenating a NAT64 prefix to its IPv4 address. When an IPv6 client queries DNS for the address of an IPv4-only resource, it receives the IPv6 address generated by the DNS64 resolver. Because you can optionally configure your DNS servers with these IPv6 addresses manually, the DNS64 resolver is not required.
- The NAT64 gateway receives the IPv6 packet whose destination IPv6 address was generated by the DNS64 resolver. It then translates the destination IPv6 address to the IPv4 address owned by the IPv4 resource. The IPv4 packet returned from the IPv4 resource is then mapped back to the destination IPv6 address from the client's request.

Requirements for stateless NAT46 configurations

The ServerIron ADX must be configured as a NAT46 gateway. The NAT46 gateway receives IPv4 packets whose destination IPv4 address is mapped to an internal IPv6 resource. It then translates the IPv4 address to an IPv6 address using the IPv4-IPv6 mapping table information. Return packets from the IPv6 resource are then mapped back to the client's IPv4 address. NAT46 is stateless, meaning that the NAT46 device does not keep track of the connections between the IPv4 client and the IPv6 resource.

Protocol support

The Foundry implementation of NAT64 and NAT46 supports the following protocols:

- Generic Transmission Control Protocol (TCP)
- Generic User Datagram Protocol (UDP)
- Hypertext Transfer Protocol (HTTP)
- HTTP Secure (HTTPS)
- Internet Control Message Protocol (ICMP)
- Remote Desktop Protocol (RDP)

NOTE
TFTP is not supported with NAT64.

NAT64 ICMPv6 to ICMPv4 message handling

NAT64 ICMP packet handling allows ICMPv6 messages to be translated to the corresponding ICMPv4 messages on the IPv4 side, and ICMPv4 messages from the IPv4 side to be translated to the corresponding ICMPv6 messages on the IPv6 side.

Table 1 shows how ICMPv6 messages are translated to ICMPv4 messages.

TABLE 1 ICMPv6 to ICMPv4 message translation

ICMPv6 message type	ICMPv4 message type
Destination Unreachable (Type 1)	Destination Unreachable (Type 3)
no route to destination (code 0)	destination host unreachable (code 1)
communication with destination administratively prohibited (code 1)	host administratively prohibited (code 10)
beyond scope of source address (code 2)	source route failed (code 5)
address unreachable (code 3)	destination host unreachable (code 1)
port unreachable (code 4)	destination port unreachable (code 3)
Packet Too Big (Type 2)	Destination Unreachable (Type 3) fragmentation required, and DF flag set (code 4)

TABLE 1 ICMPv6 to ICMPv4 message translation (Continued)

ICMPv6 message type	ICMPv4 message type
Time Exceeded (Type 3)	Time Exceeded (Type 11) Code remains same from ICMPv6
Parameter Problem (Type 4) unrecognized Next Header type encountered (code 1)	Destination Unreachable (Type 3) destination protocol unreachable (code 2)
Parameter Problem (Type 4) Any other parameter problem	Parameter problem: bad IP header (Type 12) pointer indicates the error (code 0)
Echo Request (Type 128)	Echo Request (Type 8)
Echo Reply (Type 129)	Echo Reply (Type 0)

Table 2 shows how ICMPv4 messages are translated to ICMPv6 messages.

TABLE 2 ICMPv4 to ICMPv6 message translation

ICMPv4 message type	ICMPv6 message type
Destination Unreachable (Type 3)	Destination Unreachable (Type 1)
destination network unreachable (code 0)	no route to destination (code 0)
destination host unreachable (code 1)	no route to destination (code 0)
destination protocol unreachable (code 2)	Parameter Problem (Type 4) unrecognized Next Header type encountered (code 1)
destination port unreachable (code 3)	Destination Unreachable (Type 1) port unreachable (code 4)
fragmentation required, and DF flag set (code 4)	Packet Too Big (Type 2)
source route failed (code 5)	Destination Unreachable (Type 1) not neighbor code (code 1)
destination network unknown (code 6)	no route to destination (code 0)
destination host unknown (code 7)	no route to destination (code 0)
source host isolated (code 8)	no route to destination (code 0)
network administratively prohibited (code 9)	communication with destination administratively prohibited (code 1)
host administratively prohibited (code 10)	communication with destination administratively prohibited (code 1)
network unreachable for TOS (code 11)	no route to destination (code 0)
host unreachable for TOS (code 12)	no route to destination (code 0)
communication administratively prohibited (code 13)	communication with destination administratively prohibited (code 1)

TABLE 2 ICMPv4 to ICMPv6 message translation (Continued)

ICMPv4 message type	ICMPv6 message type
host precedence violation (code 14)	communication with destination administratively prohibited (code 1)
precedence cutoff in effect (code 15)	communication with destination administratively prohibited (code 1)
Time Exceeded (Type 11)	Time Exceeded (Type 3) Code remains same from ICMPv4
Destination Unreachable (Type 3) destination protocol unreachable (code 2)	Parameter Problem (Type 4) unrecognized Next Header type encountered (code 1)
Parameter problem: bad IP header (Type 12) pointer indicates the error (code 0)	Parameter Problem (Type 4) Any other parameter problem
Echo Request (Type 8)	Echo Request (Type 128)
Echo Reply (Type 0)	Echo Reply (Type 129)

NAT64 full-sized packet handling

NAT64 full-sized packet handling enables full-sized packets from the IPv4 host to be split into multiple packets for the IPv6 client. This functionality is needed because an IPv4 header is 20 bytes in length and an IPv6 header (along with extension headers if applicable) is greater than or equal to 40 bytes.

If the ServerIron ADX receives a full-sized packet (total length of IPv4 packet equal to the MTU) on the IPv4 side, when it translates the IPv4 header to the IPv6 header, the total length of the IP packet may exceed the MTU of the IPv6 side, so this full-sized packet must be split into two IPv6 fragments before being sent to the IPv6 host. The head fragment size will be equal to the IPv6-side MTU and the non-head fragment size will be equal to the original packet size minus the first fragment size.

NAT64 fragmentation support

NAT64 fragmentation handling enables the ServerIron ADX to convert an IPv4 fragmented packet to an IPv6 fragmented packet and vice versa. The fragmentation formats are different for IPv4 and IPv6.

For the IPv6 to IPv4 direction, the ServerIron ADX:

- Strips off the IPv6 fragment ed header
- Extracts the information stored in it
- Populates the fragment ID, offset, and flags of the IPv4 header

For the IPv4 to IPv6 direction, the ServerIron ADX:

- Extracts information stored in the IPv4 header
- Creates and populates the IPv6 fragmented header

NOTE
Because the ICMP checksum mechanism in IPv6 is different than the IPv4 checksum mechanism, ICMP fragmentation is currently not supported, and all fragmented ICMP packets received on either the IPv6 or IPv4 side are dropped. Counters keep track of the number of fragmented ICMP packets dropped.

NOTE
When an IPv4 host sends multiple fragments with UDP checksum 0, the translation of those packets from IPv4 to IPv6 is not supported.

Installing NAT64 on a previously configured ServerIron ADX

If a ServerIron ADX was previously configured for server load balancing (SLB), any server configuration statements must be removed prior to creating any NAT64 configuration, and the device must be rebooted.

This can be done by either of the following methods:

- Option 1: Use the **no** form of each complete server command to remove each configuration statement or section in a granular fashion. Write the configuration to memory and then reload.
- Option 2: Completely erase the entire configuration with the **erase startup-config** command and then reload without saving the configuration.

References

- RFC 6144: Framework for IPv4/IPv6 Translation
- RFC 6145: IP/ICMP Translation Algorithm
- RFC 6146: Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers
- RFC 6147: DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers

Chapter

2

Stateful NAT64 Configuration

In this chapter

- Stateful NAT64 overview 7
- Basic stateful NAT64 configuration 9
- Advanced stateful NAT64 configuration. 11
- High availability for stateful NAT64 18
- Displaying NAT64 information 19
- Clearing stateful NAT64 information 26
- Debugging stateful NAT64 configurations 26

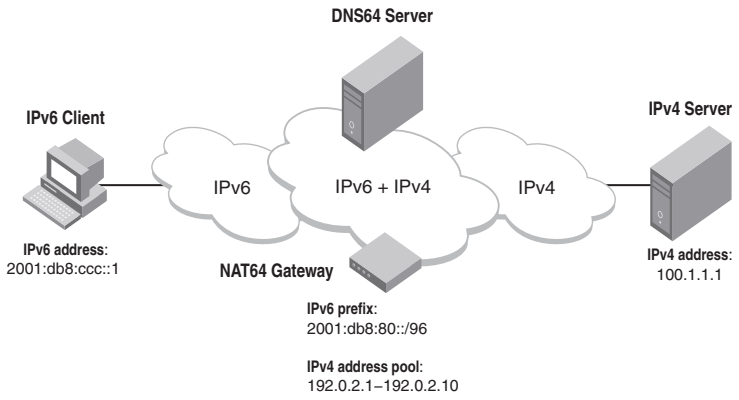
Stateful NAT64 overview

In a stateful NAT64 configuration, a ServerIron ADX configured as a NAT64 gateway enables IPv6-only clients and IPv4 resources to communicate with each other by way of address and packet translation. This translation operation is performed on the NAT64 gateway using stateful sessions.

Stateful NAT64 components

Figure 1 provides a high-level view of a sample stateful NAT64 configuration. As shown, the client resides in an IPv6-only network and the server (resource) resides in an IPv4-only network. The DNS64 server and the NAT64 gateway communicate with both the IPv4 and IPv6 networks.

FIGURE 1 IPv6-only client to IPv4 resource overview



The DNS64 server provides the IPv6 client with a synthesized IPv6 address which enables the IPv6 client to reach the IPv4 resource. The synthesized IPv6 address consists of the 96-bit NAT64 IPv6 prefix concatenated to the IPv4 destination address of the IPv4 resource and represents that IPv4 resource to the IPv6 network.

The ServerIron ADX is configured as a stateful NAT64 gateway. The NAT64 gateway strips the IPv6 prefix from the synthesized destination address and forwards the packet to the resource using an IPv4 address selected from its IPv4 NAT address pool as the source address. Because the state is maintained, packets returned from the IPv4-only resource to the IPv6-only client are able to use the same session to perform the required reverse translation.

Two key components must be configured on the NAT64 gateway in order to allow communication between IPv6 clients and IPv4 resources: a NAT64 IPv6 prefix and an IPv4 NAT address pool.

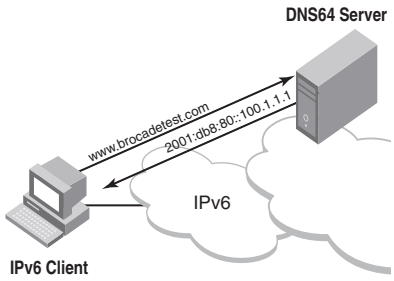
- NAT64 IPv6 prefix: The DNS64 server concatenates the NAT64 IPv6 prefix to the IPv4 source address to create the synthesized IPv6 destination address which represents the IPv4 resource to the IPv6 network.
- IPv4 NAT address pool: The NAT64 gateway selects an IPv4 address from the IPv4 NAT address pool to use as the source address for packets it forwards to the IPv4 resource.

Operation of stateful NAT64

The following steps describe the operation of stateful NAT64 translation for the environment shown in Figure 1.

1. The IPv6 client sends a query to the DNS64 server for the address of the resource “www.brocadetest.com”.
2. As shown in Figure 2, the DNS64 server responds to the client’s query with a synthesized IPv6 address (2001:db8:80::100.1.1.1) that represents the IPv4 resource to the IPv6 network. Notice that this synthesized IPv6 address is composed of a user-configured 96-bit IPv6 prefix (2001:db8:80::) and the IP address of the IPv4-only server (100.1.1.1).

FIGURE 2 IPv6 client to DNS64 server communication



3. The IPv6 client sends a packet with the source address of the IPv6 client (2001:db8:ccc::1) to the synthesized IPv6 destination address (2001:db8:80::100.1.1.1) that was obtained from the DNS64 server.
4. The NAT64 gateway strips away the NAT64 IPv6 prefix from the synthesized IPv6 destination address that was sent from the IPv6-only client and sets the remaining IPv4 address (100.1.1.1) as the new destination address.

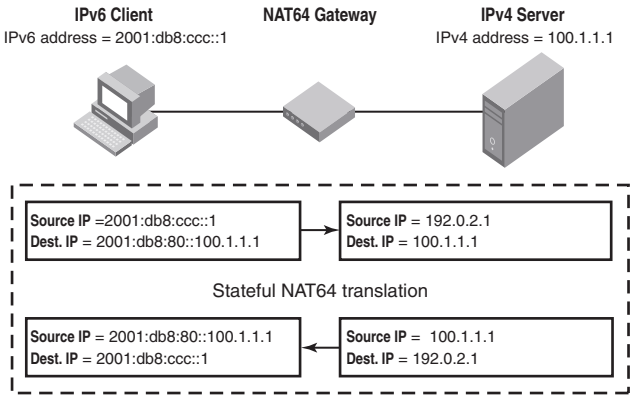
5. The NAT64 gateway also dynamically selects an IPv4 address (192.0.2.1) from the IPv4 NAT address pool (192.0.2.1–192.0.2.10) and uses that address as the source address for the packet that is sent to the IPv4 resource.

   In a stateful configuration, the NAT64 gateway keeps track of all its connections.

6. When a return packet destined for the IPv6 client arrives at the NAT64 gateway, the NAT64 gateway translates that packet's IPv4 destination address (192.0.2.1) to the client's IPv6 address (2001:db8:ccc::1) and the IPv4 resource's source address (100.1.1.1) to the synthesized IPv6 address (2001:db8:80::100.1.1.1).

Figure 3 illustrates stateful NAT64 translation.

FIGURE 3 Stateful NAT64 translation



NOTE
If the ServerIron ADX receives an IPv6 packet that contains a protocol other than TCP, UDP, or ICMPv6 in the last Next Header field of the IPv6 header, then the packet is discarded silently.

Basic stateful NAT64 configuration

The steps required for basic stateful NAT64 configuration include the following:

- "Configuring stateful IPv6 prefixes" on page 9
- "Configuring basic IPv4 NAT address pools" on page 10

Advanced configuration options are described later in the chapter.

NOTE
If you are configuring NAT64 on a system that has had a previous SLB configuration, refer to "Installing NAT64 on a previously configured ServerIron ADX" on page 6.

Configuring stateful IPv6 prefixes

In a stateful NAT64 configuration, the DNS64 server uses an IPv6 prefix to create a synthesized IPv6 address which represents an IPv4 resource to the IPv6 network. This synthesized IPv6 address consists of two parts: an IPv6 prefix and the IPv4 address of an IPv4 resource.

To specify an IPv6 prefix, enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::/96
```

Syntax: [no] **nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route**]

NOTE
A maximum of eight NAT64 IPv6 prefixes can be configured.

The <*prefix/length*> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway. Currently, the only supported prefix length is 96.

The **inject-static-route** option is used only in advanced configurations. For advanced configurations, refer to "Configuring stateful IPv6 prefixes with route injection" on page 12.

Configuring basic IPv4 NAT address pools

The IPv4 NAT address pool specifies a range of IPv4 addresses that the NAT64 gateway can use to represent an IPv6 client to the IPv4 network.

During stateful NAT64 translation, the NAT64 gateway substitutes an IPv4 address selected from the IPv4 NAT address pool for the actual IPv6 source address of the IPv6 client.

To configure an IPv4 NAT address pool, enter a command such as the following:

```
ServerIron ADX(config)# nat64 pool nat64-zone1 201.1.1.1 201.1.1.20 prefix-length
24
```

In the example, an IPv4 NAT address pool named "nat64-zone1" is created on the NAT64 gateway consisting of addresses between 201.1.1.1 and 201.1.1.20.

Syntax: [**no**] **nat64 pool** <*pool-name*> <*start-ipaddress*> <*end-ipaddress*> **prefix-length** <*prefix-length*> [**port-pool-range** { **1** | **2** }] [**inject-static-route**]

NOTE
A maximum of 192 addresses can be configured for the IPv4 NAT address pools.

The <*pool-name*> variable specifies the name of the IPv4 NAT address pool. The name can be up to 255 characters in length.

The <*start-ipaddress*> variable specifies the IPv4 address at the beginning of the NAT64 pool range. This value should be the lowest numbered IPv4 address in the range.

The <*end-ipaddress*> variable specifies the IPv4 address at the end of the IPv4 NAT address pool range. This value should be the highest numbered IPv4 address in the range.

The **prefix-length** option uses the CIDR prefix value specified by the <*prefix-length*> variable to distinguish the portion of the IPv4 address that will be used for all IPv4 addresses in the pool. For example, you can use "24" for the length of the CIDR prefix.

The **port-pool-range** option enables NAT64 redundancy. A value of **2** specifies a higher priority of the NAT IP address than a value of **1**. A value of **2** also indicates that source ports allocated for the NAT IP address are from the higher range.

The **inject-static-route** option is used only in advanced configurations. For advanced configurations, refer to "Configuring IPv4 NAT address pools with route injection" on page 13.

NOTE
If the ServerIron ADX runs out of stateful NAT64 NAT pool ports or session entries, new connection requests are dropped silently.

NOTE
You must reload the ServerIron ADX whenever the IPv4 NAT address pool configuration is changed.

Advanced stateful NAT64 configuration

An advanced stateful NAT64 configuration includes one or more optional features in addition to the basic configurations.

The steps used for advanced stateful NAT64 configuration include the following:

- "Configuring stateful NAT64 with route injection" on page 11
- "NAT64 sticky session configuration" on page 15
- "Enabling connection logging" on page 16
- "Configuring HTTP client IP address insertion" on page 16
- "Configuring NAT64 packet fragmentation options" on page 17

Configuring stateful NAT64 with route injection

The NAT64 gateway uses route injection to advertise IPv6 and IPv4 address translations to the appropriate networks. Using static route injection, you can make these translations available as destinations in the routing tables of the IPv6 and IPv4 networks in which they reside. Route distribution is then performed using any of the routing protocols supported by the ServerIron ADX: OSPF, IS-IS, and BGP (IPv4 and IPv6).

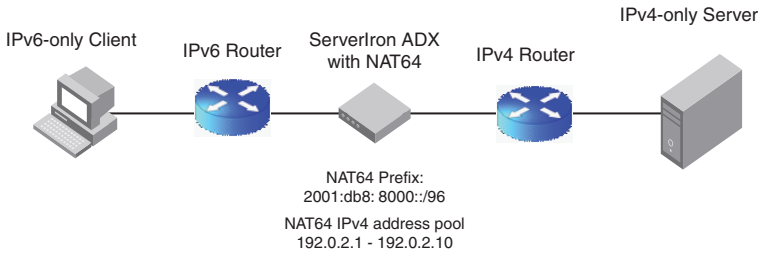
- Synthesized NAT64 IPv6 addresses that represent IPv4-only resources to the IPv6 network can be injected into the routing table for the IPv6 network.
- IPv4 addresses (selected from the IPv4 NAT address pool) that represent IPv6 clients to the IPv4 network can be injected into the routing table for the IPv4 network.

NAT64 route injection requires that the ServerIron ADX have a routing adjacency relationship with a router for the protocols for which you want to support route distribution.

- For the IPv4 NAT address pool, the **nat64 pool** command has the **inject-static-route** option which injects a route to the subnet into the local routing table. This route is then advertised as the destination for traffic bound for the subnet by the routing protocol that is configured on the ServerIron ADX.
- Starting with ServerIron ADX release 12.4.00, configuring the **inject-static-route** option with the **nat64 ipv6-prefix** command allows the configured IPv6 routing protocol to advertise the subnet defined by the IPv6 prefix to the adjacent IPv6 neighbor and thereby make it available in the routing tables of the IPv6 network, The next hop to the subnet is advertised to the IPv6 network as the adjacent router. In all prior releases of the ServerIron ADX, you are required specify an IPv6 interface on the ServerIron ADX (Ethernet or VE) that is directly connected to the adjacent router.

Figure 4 shows a typical IPv6-only client to IPv4 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX. In this configuration, routes defined by the IPv6 prefix and IPv4 NAT address pool are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured.

FIGURE 4 NAT64 route injection



NOTE
For details about how to configure routing protocols on a ServerIron ADX, refer to the following chapters in the *ServerIron ADX Switch and Router Guide*: “Configuring OSPF”, “Configuring IPv6 Dynamic Routing”, “Configuring IS-IS (IPv4)”, “Configuring IPv6 IS-IS”, “Configuring BGP4 (IPv4)”, and “Configuring BGP4+”.

Configuring stateful IPv6 prefixes with route injection

In a stateful NAT64 configuration, the DNS64 server uses an IPv6 prefix to create a synthesized IPv6 address which represents an IPv4 resource to the IPv6 network. This synthesized IPv6 address consists of two parts: an IPv6 prefix and the actual IPv4 address of an IPv4 resource.

Advanced NAT64 configurations can use static route injection to inject a route to IPv4 resources that are mapped to the subnet defined by the IPv6 prefix into an IPv6 routing table, which is then advertised to the adjacent IPv6 router.

To specify an IPv6 prefix and a route to the subnet defined by that prefix, enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::/96 inject-static-route
```

In the example, an IPv6 prefix (2001:db8:8000::) is defined that specifies the IPv6 subnet. The **inject-static-route** option is used to specify the route to this subnet that will be injected into the local routing table.

Syntax: [no] nat64 ipv6-prefix <*prefix/length*> [**inject-static-route**]

The <*prefix/length*> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway. Currently, the only supported prefix length is 96.

NOTE
A maximum of eight NAT64 IPv6 prefixes can be configured.

The **inject-static-route** option is used to advertise the subnet defined by the <*prefix/length*> variable to the IPv6 network.

In ServerIron ADX releases prior to 12.4.00, you must also identify either an Ethernet or VE interface and port number on the NAT64 gateway for static route injection. The specified interface must have an IPv6 address and be directly connected to an adjacent router.

Syntax: **[no] nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route** {**ve** <*port-number*> | **ethernet** <*port-number*>}]

The **ve** <*port-number*> or **ethernet** <*port-number*> variable specifies an IPv6 interface that is connected to the adjacent IPv6 router.

The **inject-static-route** option is described in detail in "Configuring stateful NAT64 with route injection" on page 11.

Configuring IPv4 NAT address pools with route injection

In advanced configurations, you can use the **inject-static-route** option with the **nat64 pool** command to ensure that a route to the specified IPv4 subnet is injected into the local routing table. This route is then advertised as the destination for traffic bound for the subnet by the routing protocol that is configured on the ServerIron ADX.

To configure a NAT64 IPv4 address pool and specify a route to the IPv4 subnet, enter a command such as the following:

```
ServerIron ADX(config)# nat64 pool nat64-zone1 201.1.1.1 201.1.1.20 prefix-length
24 inject-static-route
```

In the example, the **inject-static-route** option is used to ensure that the pool of IPv4 addresses used to represent the IPv6 clients is made available on the IPv4 network.

Syntax: [**no**] **nat64 pool** <*pool-name*> <*start-ipaddress*> <*end-ipaddress*> **prefix-length** <*prefix-length*> [**port-pool-range** { **1** | **2** }] [**inject-static-route**]

NOTE
A maximum of 192 addresses can be configured for the IPv4 NAT address pools.

The <*pool-name*> variable specifies the name of the IPv4 NAT address pool. The name of the pool can be up to 255 characters in length.

The <*start-ipaddress*> variable specifies the IPv4 address at the beginning of the IPv4 NAT address pool range. This value should be the lowest numbered IPv4 address in the range.

The <*end-ipaddress*> variable specifies the IPv4 address at the end of the IPv4 NAT address pool range. This value should be the highest numbered IPv4 address in the range.

The **prefix-length** option uses the CIDR prefix value specified by the <*prefix-length*> variable to distinguish the portion of the IPv4 address that will be used for all IPv4 addresses in the pool. For example, you can use "24" for the length of the CIDR prefix.

The **port-pool-range** option enables NAT64 redundancy. A value of **2** specifies a higher priority of the NAT IP address than a value of **1**. A value of **2** also indicates that source ports allocated for the NAT IP address are from the higher range.

The **inject-static-route** option injects the <*start-ipaddress*> /<*prefix-length*> subnet route into the routing table, which is then advertised to the adjacent router. This option is described in detail in "Configuring stateful NAT64 with route injection" on page 11.

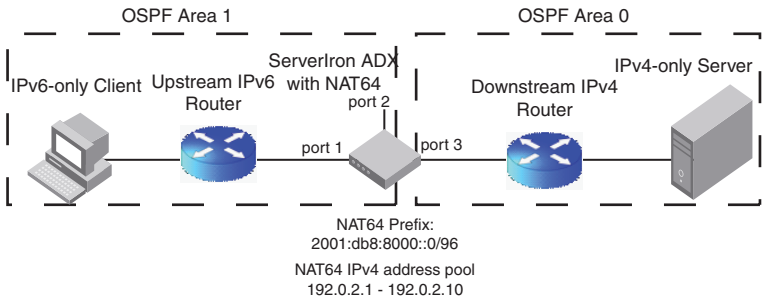
NOTE
If the ServerIron ADX runs out of stateful NAT64 NAT pool ports or session entries, new connection requests are dropped silently.

NOTE
You must reload your ServerIron ADX when the IPv4 NAT address pool configuration is changed.

Stateful NAT64 static route injection configuration example

Figure 5 shows a typical IPv6-only client to IPv4 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX. In this configuration, routes defined by the IPv6 prefix and IPv4 NAT address pool are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured.

FIGURE 5 NAT64 route injection example



The following example configures the NAT64 gateway for route injection as shown in Figure 5.

1. An IPv6 address is configured: VLAN 10 is configured with VE 10, which has an IPv6 address (2001:db8:7000::1).

```
ServerIron ADX(config)# vlan 10
ServerIron ADX(config-vlan-10)# untagged ethernet 1
ServerIron ADX(config-vlan-10)# router-interface ve 10
ServerIron ADX(config-vlan-10)# exit
ServerIron ADX(config)# interface ve 10
ServerIron ADX(config-vif-10)# ipv6 address 2001:db8:7000::1/64
ServerIron ADX(config-vif-10)# exit
```

2. An IPv4 address is configured: VLAN 20 is configured with VE 20, which has an IP address (192.168.1.1).

```
ServerIron ADX(config)# vlan 20
ServerIron ADX(config-vlan-20)# untagged ethernet 3
ServerIron ADX(config-vlan-20)# router-interface ve 20
ServerIron ADX(config-vlan-20)# exit
ServerIron ADX(config)# interface ve 20
ServerIron ADX(config-vif-20)# ip address 192.168.1.1/24
ServerIron ADX(config-vif-20)# exit
```

3. OSPF and OSPFv6 are configured for static route redistribution. The IPv4 side is configured as OSPF Area 0. The IPv6 side is configured as OSPF Area 1.

```
ServerIron ADX(config)# router ospf
ServerIron ADX(config-ospf-router)# redistribution static
ServerIron ADX(config-ospf-router)# area 0
ServerIron ADX(config-ospf-router)# exit
ServerIron ADX(config)# ipv6 router ospf
ServerIron ADX(config-ospf6-router)# redistribute static
ServerIron ADX(config-ospf6-router)# area 1
ServerIron ADX(config-ospf6-router)# exit
```

4. Assign the VE interfaces to OSPF or OSPFv6 areas defined previously.

```
ServerIron ADX(config)# interface ve 10
ServerIron ADX(config-vif-10)# ipv6 ospf area 1
ServerIron ADX(config-vif-10)# exit
ServerIron ADX(config)# interface ve 20
ServerIron ADX(config-vif-20)# ip ospf area 0
ServerIron ADX(config-vif-20)# exit
```

5. An IPv4 NAT address pool is configured with static route injection.

```
ServerIron ADX(config)# nat64 pool nat64-zone1 192.0.2.1 192.0.2.10
prefix-length 24 port-pool-range 1 inject-static-route
```

6. An IPv6 prefix is configured with static route injection.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96
inject-static-route
```

   If you are running a ServerIron ADX build prior to 12.4.00, you must specify an interface and port number.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96
inject-static-route ve 10
```

NOTE
While this example describes NAT64 route injection for the OSPF protocol, you can also use this feature with BGP and IS-IS. Only the routing protocol configurations will differ.

NAT64 sticky session configuration

This section describes steps required to configure NAT64 sticky sessions.

Enabling ServerIron ADX to delete sticky sessions

By default, the stateful NAT64 gateway selects the same NAT64 pool IP address for a given IPv6 client and maintains this IPv6 client-to-NAT64 address mapping by means of a sticky session.

The same NAT64 pool IP address is selected for all subsequent flows from the client for as long as this sticky session exists, However, under certain heavy traffic conditions, the NAT64 pool might run out of ports. In such an event, new connections from the same client are dropped by the NAT64 gateway.

To override this behavior, configure the **nat64 nat-sticky-delete** command to enable the NAT64 gateway to delete an existing sticky session that has run out of available ports and select a different NAT64 pool IP address for the same client.

```
ServerIron ADX(config)# nat64 nat-sticky-delete
```

Syntax: [no] **nat64 nat-sticky-delete**

Once this command is configured, the NAT64 gateway will automatically delete an existing sticky session for a NAT64 pool if a connection request arrives from the same IPv6 client and the NAT64 pool IP address in the associated sticky session has run out of available source ports. The NAT64 gateway then selects a new IPv4 address from the configured NAT64 pool address range and creates a new sticky session for the IPv6 client to NAT64 pool address.

Although existing sessions from the IPv6 client continue, all new connections will use the newly created sticky session.

Disabling sticky behavior

By default, the stateful NAT64 gateway selects the same NAT64 pool IP address for a given IPv6 client and maintains this IPv6 client-to-NAT64 address mapping by means of a sticky session.

For as long as this sticky session exists, the same NAT64 pool IP address is selected for all subsequent flows from the client. However, under certain heavy traffic conditions, the NAT64 pool might run out of ports. In such an event, new connections from the same client are dropped by the NAT64 gateway.

Use the **nat64 disable-sticky** command to disable sticky session behavior on the NAT64 gateway and to ensure that a new IPv4 address is selected from the IPv4 NAT address pool whenever a request comes from an IPv6 client.

```
ServerIron ADX(config)# nat64 disable-sticky
```

Syntax: [no] **nat64 disable-sticky**

Enabling connection logging

To enable connection logging for NAT64 traffic processed by the NAT64 gateway, enter the **nat64 connection-log** command.

```
ServerIron ADX(config)# nat64 connection-log
```

Syntax: [no] **nat64 connection-log**

Configuring HTTP client IP address insertion

When an IPv6 address is translated, the internal resource cannot see the client's original IPv6 address. In situations where an HTTP client's identity has to be maintained, the NAT64 gateway can be configured to insert the client IP address as an HTTP header. The header will be inserted as the last header in the HTTP request.

For example, to enable client IP insertion for HTTP traffic arriving on port 80, configure the **nat64 http-client-ip-insertion port** command as in the following example:

```
ServerIron ADX(config)# nat64 http-client-ip-insertion port 80
For example, where the original HTTP request is:
    GET /abc/index.html HTTP 1/0\r\n
    Host: foo.com\r\n
    …
    Connection: Keep-Alive\r\n
    \r\n

After insertion, the HTTP request will be:
    GET /abc/index.html HTTP 1/0\r\n
```

```
Host: foo.com\r\n
…
Connection: Keep-Alive\r\n
X-Forwarded-For: 2001:db8::6401:101\r\n
\r\n
```

NOTE
Client IP address insertion must be enabled for the port handling HTTP traffic. The ServerIron ADX will not automatically detect HTTP traffic on any port.

NOTE
A client IP address will only be inserted for the first HTTP request in a TCP connection. This means that for a keep-alive or pipelined request, the client IP address header is inserted for the first request but not for subsequent requests.

Syntax: [**no**] **nat64 http-client-ip-insertion** {**port** <*port*> | **acl** <*acl-name*>} <*header-string*>

The **port** option enables client IP insertion for the <*port*> specified.

The **acl** option enables you to direct client IP insertion as defined by the ACL specified in the <*acl-name*> variable.

The <*header-string*> variable enables you to specify a customized header. If no header is specified, the default header of "X-Forwarded-For" is used.

To specify a different header name, enter a command such as the following:

```
ServerIron ADX(config)# nat64 http-client-ip-insertion port 80 "NAT64-CLIENT-IP"
```

In the example, the ServerIron ADX is configured to insert "NAT64-CLIENT-IP" as the header in requests to an IPv4 server rather than the default header of "X-Forwarded-For".

Configuring NAT64 packet fragmentation options

Reverse packets from the IPv4 server to the IPv6 client can be too large and must be split into two IPv6 packets. The following describes the criteria for judging that packets are too large:

- Regular packets: IP packet total length greater than 1480 bytes
- Fragmented packets: IP packet total length greater than 1480 + 8 bytes

If the packets exceed these limitations, one of the following actions will be taken:

1. If the **ipv6 frag-full-4to6** command is configured, the packet will be split and no further actions will be performed.
2. If the condition in step 1 is not met, and the DF bit is set at the server, the "fragmentation needed" ICMP error message will be sent.
3. If the conditions in steps 1 and 2 are not met, the packet will be split.

The **ipv6 frag-full-4to6** command is configured as shown in the following example.

```
ServerIronADX(config)# ipv6 frag-full-4to6
```

Syntax: [**no**] **ipv6 frag-full-4to6**

For more information about NAT64 fragmentation support, refer to "NAT64 fragmentation support" on page 5.

NOTE
The **ipv6-frag-full-4to6** command was introduced in ServerIron ADX release 12.4.00 and it replaces the **frag-664-reverse-full-sized-pkt** command from earlier releases.

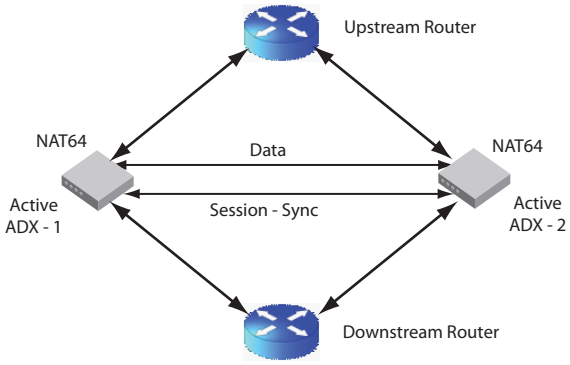
If the IPv4 host sends out fragmented UDP packets with checksum 0, the translation of those packets from IPv4 to IPv6 is not supported.

High availability for stateful NAT64

The ServerIron ADX only supports Active-Active high availability (HA) mode with NAT64. In Active-Active HA mode, the NAT64 IPv4 NAT pool ports for each NAT IP address are divided equally among ServerIron ADX HA partners. This feature is available only for a ServerIron ADX running router code.

Figure 6 shows a sample NAT64 configuration in which each ServerIron ADX is configured with a NAT64 prefix and an IPv4 NAT pool with the port-pool-range option that specifies the IPv4 NAT pool port ownerships. Sessions are synchronized periodically between the ServerIron ADX HA partners using a separate link.

FIGURE 6 NAT64 Active-Active HA configuration



NOTE
Fragmented traffic sessions do not sync in HA mode.

Configuring the port pool range for HA

When Active-Active HA is configured on a ServerIron ADX, you can specify the range of source ports allocated for the NAT IP address for each of the NAT64 gateways configured within the HA setup using the **nat64 pool** command with the **port-pool-range** option.

The **port-pool-range** option enables NAT64 redundancy. A value of **2** specifies a higher priority of the NAT IP address then a value of **1**. A value of **2** also indicates that source ports allocated for the NAT IP address are from the higher range.

For example, the following command configures an IPv4 NAT address pool with the **port-pool-range** option set to a value of **2**.

```
ServerIron ADX(config)# nat64 pool nat64-zone1 201.1.1.1 201.1.1.20 prefix-length
24 port-pool-range 2
```

NOTE
Active-active NAT configurations use the **server active-active-port ethernet** <*portnum*> **vlan-id** <*vlan-number*> command to identify the port that connects the ServerIron ADX to its Active-Active partner. The port you specify must be in its own port-based VLAN. The configuration is mandatory on both HA boxes. For more information, refer to "Configuring VLAN option for Active-Active links" in the *ServerIron ADX Server Load Balancing Guide*.

For details on configuring the **nat64 pool** command, refer to "Configuring basic IPv4 NAT address pools" on page 10.

Enabling inject active only for HA

The **nat64 inject-active-only** command ensures that only an active ServerIron ADX will inject the static route and the standby ServerIron ADX will withdraw the static routes. Without this command (default configuration), both the active and standby ServerIron ADXs will inject the route.

```
ServerIron ADX(config)# nat64 inject-active-only
```

Syntax: [**no**] **nat64 inject-active-only**

NOTE
This command only applies to the IPv6-only client to IPv4 resource configuration as described in this chapter because it is performed in the stateful mode and maintains session relationships.

Displaying NAT64 information

This section includes the following topics, which document the show commands for displaying information about stateful NAT64 configurations:

- "Displaying session information" on page 19
- "Displaying NAT64 translations" on page 20
- "Displaying NAT64 statistics" on page 21
- "Displaying NAT64 resources" on page 25

NOTE
Many of these commands can also be used to view information about stateless NAT64 and stateless NAT46 configurations as well.

Displaying session information

Use the **show session all** command at the rconsole to display sessions on the ServerIron ADX including stateful NAT64 sessions. Sessions are not created for stateless NAT64 or stateless NAT46 and the command cannot be used for stateless configurations.

NAT64 sessions are indicated by a unique session type in the output. This output is displayed as follows.

```
ServerIron ADX# rconsole 1 1
```

```
ServerIron ADX1/1 show session all 0
Session Info:
Flags - 0:UDP, 1:TCP, 2:IP, 3:INT, 4:INVD, H: sessInHash, N: sessInNextEntry
Index Src-IP          Dst-IP            S-port  D-port Age  Server   Flags
===== ======          ======            ======  ====== ===  ======   ========
0     192.168.1.101   200.1.1.2         80      38912  60   n/a      NAT641 H
1     3003::10        0.0.0.0           0       0      57   n/a      NAT641 H
2     3003::10        4003::c0a8:165    1417    80     60   n/a      NAT641 H
```

Syntax: show session all <*index*>

The <*index*> variable specifies the integer value of the record in the table from which you want to begin the display. Using the value "0" will display all records in the table.

Table 3 describes the fields returned by the **show session all** command.

TABLE 3 Display fields for show session all

Field	Description
Index	The row number of this entry in the Session Table.
Src-IP	The source IP address for the session flow.
Dst-IP	The destination IP address for the session flow.
S-port	The Layer 4 source port number for the session flow.
D-port	The Layer 4 destination port number for the session flow.
Age	The session age.
Server	The real server name the session belongs to. N/A to NAT64 sessions.
Flags	Flag identifying session type. For example, NAT64 TCP sessions are identified as NAT641. Flags displayed can be: 0:UDP 1:TCP 2:IP 3:INT 4:INVD H: sessInHash N: sessInNextEntry

Displaying NAT64 translations

You can use the **show nat64 translation** command to display the client IP, NAT IP, and destination IP addresses in a stateful NAT64 gateway. This output is displayed as follows.

```
ServerIron ADX# show nat64 translation
Pro  Client IP          NAT IP             Dest IP
tcp  3003::10|1418      200.1.1.2|38913    192.168.1.101|80
```

Syntax: show nat64 translation

Table 4 describes the fields returned by the **show nat64 translation** command.

TABLE 4 Display fields for show nat translation

Field	Description
Pro	The Layer 4 protocol: TCP. UDP or ICMP.
Client IP	The IPv6 client IP address.
NAT IP	The translated IP address from the NAT64 IPv4 pool.
Dest IP	The IP address of the internal IPv4 resource.

Displaying NAT64 statistics

You can use the **show nat64 statistics** command to display statistics for the NAT64 gateway. This command can be used for stateful NAT64, stateless NAT64, and stateless NAT46 configurations.

The following output is for a stateful NAT64 configuration.

```
ServerIron ADX# rconsole 1 1
ServerIron ADX1/1# show nat64 statistics all
*********************
NAT64 Gateway STATISTICS
*********************
         Packet processing:
         Pre-proc internal SIP map create = 18
         Pre-proc create DIP by removing prefix = 17
         Pre-proc pending drop = 0
         Stateless IPv6 prefix prepended = 0
         Stateful IPv6 prefix prepended = 31
         6->4 initiate dynamic learning = 0
         4->6 initiate dynamic learning = 0
         6->4 cannot initiate learning table full = 0
         4->6 cannot initiate learning table full = 0

         IPv6 statistics:
         6->4 pkts recv = 18
         6->4 v6 error = 0

         Fast path statistics:
         6->4 fast path candidate = 18
         6->4 fast path processed pkt = 17
         6->4 fast path processed pkt last sec = 0
         6->4 fast path processed bytes = 2235
         6->4 fast path processed bytes last sec = 0
         6->4 fast path num tcp session lookup = 17
         6->4 fast path num tcp session lookup found = 14
         4->6 fast path candidate = 16
         4->6 fast path processed pkt = 16
         4->6 fast path processed pkt last sec = 0
         4->6 fast path processed bytes = 1371
         4->6 fast path processed bytes last sec = 0
         4->6 nat64 num tcp session lookup = 16
         4->6 fast path nat64 num tcp session lookup found = 0

         Slow path statistics:
         Slow path processed pkt = 0

         Stateless Statistics:
         TCP 6->4 = 0
         TCP 4->6 = 0
```

```
UDP 6->4 = 0
UDP 4->6 = 0
Static pending or error in entry drop = 0

Stateful Statistics:
TCP 6->4 = 17
TCP 4->6 = 16
TCP reverse no session drop = 0
UDP 6->4 = 0
UDP 4->6 = 0
UDP reverse no session drop = 0

NAT64 NAT pool Statistics:
6->4: TCP: NAT port allocated = 3
6->4: TCP: NAT port not available = 0
NAT port freed = 3

NAT64 HA Statistics:
Message sync sent = 0
Message sync received = 0
Error during sending sync messages = 0
Error during receiving sync messages = 0
Invalid prefix index found during message syncing = 0

Errors:
IPv6 Prefix index invalid during pre-pend = 0
Static map entry null in NAT64 proc = 0
Get MTU on V6 outgoing port error = 0

Abnormal packets:
Embedded IPv4 address not unicast = 0
SIP contains IPv6 prefix = 0
```

The command may be run at either the EXEC CLI level or at the rconsole. The command syntax is slightly different depending on the interface used.

Syntax: show nat64 statistics [**all** | **stateless** | **dns-dynamic-learning**]

The **all** parameter displays all NAT64 statistics. Use this parameter to view stateful NAT64 statistics. The **all** parameter is available in both EXEC mode and at the rconsole.

The **stateless** parameter displays stateless NAT64 or NAT46 statistics only. The **stateless** parameter is available in both EXEC mode and at the rconsole.

The **dns-dynamic-learning** parameter displays DNS dynamic learning statistics. The **dns-dynamic-learning** parameter is available in EXEC mode only.

Table 5 describes the fields returned by the **show nat64 statistics** command.

TABLE 5 Display fields for show nat64 statistics

Field	Description
Packet processing:	
Pre-proc internal SIP map create =	INTERNAL source IP mappings created.
Pre-proc create DIP by removing prefix =	NAT64 IPv4 destination IP addresses extracted.
Pre-proc pending drop =	NAT46 (STATELESS) Number of packets dropped due to pending DNS dynamic discovery.

TABLE 5 Display fields for show nat64 statistics (Continued)

Field	Description
Stateless IPv6 prefix prepended =	Stateless NAT46 IPv4->IPv6 packet conversions.
Stateful IPv6 prefix prepended =	Stateful NAT64 IPv4->IPv6 packet conversions.
6->4 initiate dynamic learning =	Stateless: Number of DNS dynamic learnings initiated to discover IPv4 address.
4->6 initiate dynamic learning =	Stateless: Number of DNS dynamic learnings initiated to discover IPv6 address.
6->4 cannot initiate learning table full =	# DNS learning not initiated for IPv4 address discovery due to table being full.
4->6 cannot initiate learning table full =	# DNS learning not initiated for IPv6 address discovery due to table being full.
IPv6 statistics:	
6->4 pkts recv =	# IPv6 NAT64 packets received.
6->4 v6 error =	NAT64 IPv6 packets received with errors.
Fast path statistics:	
6->4 fast path candidate =	# IPv6 packets identified for optimized IPv4 conversion when NAT64 is the only feature enabled.
6->4 fast path processed pkt =	# IPv6 packets that underwent optimized IPv4 conversion when NAT64 is the only feature enabled.
6->4 fast path processed pkt last sec =	# of optimized IPv6 to IPv4 converted packets in the last second.
6->4 fast path processed bytes =	# bytes in IPv6 to IPv4 converted packets.
6->4 fast path processed bytes last sec =	# bytes in IPv6 to IPv4 converted packets in the last second.
6->4 fast path num tcp session lookup =	# NAT64 TCP session lookups for IPv6 to IPv4 packets.
6->4 fast path num tcp session lookup found =	# successful TCP session lookups for IPv6 to IPv4 packets.
4->6 fast path candidate =	# IPv4 packets identified for optimized IPv6 conversion when NAT64 is the only feature enabled.
4->6 fast path processed pkt =	# IPv4 packets that underwent optimized IPv6 conversion when NAT64 is the only feature enabled.
4->6 fast path processed pkt last sec =	# bytes in IPv4 to IPv6 converted packets in the last second.
4->6 fast path processed bytes =	# bytes in IPv4 to IPv6 converted packets.
4->6 fast path processed bytes last sec =	# bytes in IPv4 to IPv6 converted packets in the last second.
4->6 nat64 num tcp session lookup =	# successful TCP session lookups for IPv4 to IPv6 packets.
4->6 fast path nat64 num tcp session lookup found =	# successful TCP session lookups for IPv4 to IPv6 packets.
Slow path statistics:	
Slow path processed pkt = 0	Number of NAT64 packets that did not go through fast-path because of other features being used.
Stateless Statistics:	
TCP 6->4 =	# stateless NAT64 TCP IPv6 packets converted to IPv4.

TABLE 5 Display fields for show nat64 statistics (Continued)

Field	Description
TCP 4->6 =	# stateless NAT64 TCP IPv4 packets converted to IPv6.
UDP 6->4 =	# stateless NAT64 UDP IPv6 packets converted to IPv4.
UDP 4->6 =	# stateless NAT64 UDP IPv4 packets converted to IPv6.
Static pending or error in entry drop =	
Stateful Statistics:	
TCP 6->4 =	The number of IPv6 TCP packets that have been translated to IPv4.
TCP 4->6 =	The number of IPv4 TCP packets that have been translated to IPv6.
TCP reverse no session drop =	The number of TCP packets that were dropped due to no reverse session being present.
UDP 6->4 =	The number of IPv6 UDP packets that have been translated to IPv4.
UDP 4->6 =	The number of IPv4 UDP packets that have been translated to IPv6.
UDP reverse no session drop =	The number of UDP packets that were dropped due to no reverse session being present
NAT64 NAT pool Statistics:	
6->4: TCP: NAT port allocated =	The total number of ports allocated from the NAT64 IPv4 pool.
6->4: TCP: NAT port not available =	The total number of occurrences when no NAT64 IPv4 pool port was available.
NAT port freed =	The total number of ports allocated from NAT64 IPv4 pool that have been freed up.
NAT64 HA Statistics:	
Message sync sent =	Number of sessions synced to HA peer.
Message sync received =	Number of NAT64 sessions received from HA peer.
Error during sending sync messages =	# of errors encountered while syncing NAT64 sessions.
Error during receiving sync messages =	# of sync messages with errors (usually indicates configuration mismatch).
Invalid prefix index found during message syncing =	# of times NAT64 prefix in session received from peer not configured on device.
Errors	
IPv6 Prefix index invalid during pre-pend =	# times when IPv6 prefix was not found when converting IPv4 -> IPv6 packets.
Static map entry null in NAT64 proc =	Not used.
Get MTU on V6 outgoing port error =	Not used.
Abnormal packets:	
Embedded IPv4 address not unicast =	IPv4 address embedded in the IPv6 prefix is a multi-cast / broadcast address.
SIP contains IPv6 prefix =	Packet received has source IP in NAT64 prefix.

Displaying NAT64 resources

You can use the **show nat64 resources** command at the rconsole to display information about NAT64 resources on the ServerIron ADX. This command can be used for stateful NAT64, stateless NAT64, and stateless NAT46 configurations. This output is displayed as follows.

```
ServerIron ADX# rconsole 1 1
ServerIron ADX1/1# show nat64 resources
*********************
NAT64 Gateway RESOURCES
*********************
         NAT64 only feature enabled: Yes
         NAT64 Stateless enabled: No
         NAT64 Stateful enabled: Yes

         NAT64 IPv6 prefixes:
         -------------------
         IPv6 Prefix: 4003::
         Number of IPv6 prefixes: 1
         Stateless IPv6 prefix: Not Configured

         NAT64 Stateless:
         -------------------
         IPv6 map hash table size: 1024
         Max mapping entries: 1024
         Number of free map entries: 0


         NAT64 IPv4 NAT Pools:
         -------------------
         Number of NAT pools configured: 1
         Number of NAT pool IPs configured: 1 Maximum: 192
         Head pool: 2591b140
         Current pool: 2591b140
         nat64 pool test:
          start: 200.1.1.1 end: 200.1.1.2 len: 24 port-range: no inject: no
 Pool Mem: 2591b140 Current IP: 200.1.1.2
Ports[0]: Mem: 2591b140 Head: 1 Tail: 0 IP: 200.1.1.1 Memory: 26a2a000 Total: 7168
Free: 7167
Ports[1]: Mem: 2591b140 Head: 0 Tail: 0 IP: 200.1.1.2 Memory: 26a3b000 Total: 7168
Free: 7168
```

Syntax: show nat64 resources

Table 6 describes the fields returned by the **show nat64 resources** command.

TABLE 6 Display fields for show nat64 resources

Field	Description
NAT64 Gateway RESOURCES:	
NAT64 only feature enabled:	"Yes" if NAT64 is the only feature configured. "No" otherwise.
NAT64 Stateless enabled:	"Yes" if NAT64 stateless is configured. "No" otherwise.
NAT64 Stateful enabled: Yes	"Yes" if NAT64 stateful is configured. "No" otherwise.
NAT64 IPv6 prefixes:	Lists all NAT64 IPv6 prefixes on the system.
IPv6 Prefix:	The first IPv6 prefix on the system. More IPv6 prefixes will be listed similarly.

TABLE 6 Display fields for show nat64 resources (Continued)

Field	Description
Number of IPv6 prefixes:	Lists the total number of IPv6 prefixes on the system max: 8
Stateless IPv6 prefix:	Indicates whether the IPv6 Prefix is configured as stateless.
NAT64 Stateless:	Lists all NAT46 configuration.
IPv6 map hash table size:	IPv6 map hash table size.
Max mapping entries:	The max mapping entries allowed on the system.
Number of free map entries:	The number of free map entries.
NAT64 IPv4 NAT Pools:	Lists all NAT64 pool configuration.
Number of NAT pools configured:	The number of NAT pools configured on the system.
Number of NAT pool IPs configured:	The number of NAT pool IP addresses configured on the system. The maximum number is 192.
Head pool:	Identifier of the first pool configured.
Current pool:	Identifier of the current IPv4 pool being used.
nat64 pool test:	Lists details about the NAT64 IPv4 pools on the system.

Clearing stateful NAT64 information

Use the **clear nat64 statistics** command to clear stateful NAT64 statistics on a ServerIron ADX. This command is issued as follows:

```
ServerIron ADX# clear nat64 statistics
```

Syntax: clear nat64 statistics

Debugging stateful NAT64 configurations

Use the **debug nat64** command to enable diagnostic debugging of stateful NAT64 configurations.

To debug a stateful NAT64 configuration, you must use the **stateful** operand such as in the following example.

```
ServerIron ADX# debug nat64 stateful 192.0.2.1
```

In the example, diagnostic debugging is enabled for the IPv4 resource at the IP address 192.0.2.1.

Syntax: debug nat64 stateful all | <*ipv4-address*> | <*ipv6-address*>

The **all** operand displays debug information for all IPv4 and IPv6 source and destination IP addresses.

The <*ipv4-address*> variable displays debug information for a specific IPv4 source or destination IP address.

The <*ipv6-address*> variable displays debug information for a specific IPv6 source or destination IP address.

Use the **undebug** command to disable debugging.

Chapter

3

Stateless NAT64 Configuration

In this chapter

- Stateless NAT64 overview 27
- Stateless NAT64 static mapping configuration 30
- Stateless NAT64 dynamic mapping configuration 36
- High availability for NAT64 40
- Displaying NAT64 information 40
- Clearing NAT64 information 41
- Debugging stateless NAT64 configurations 42

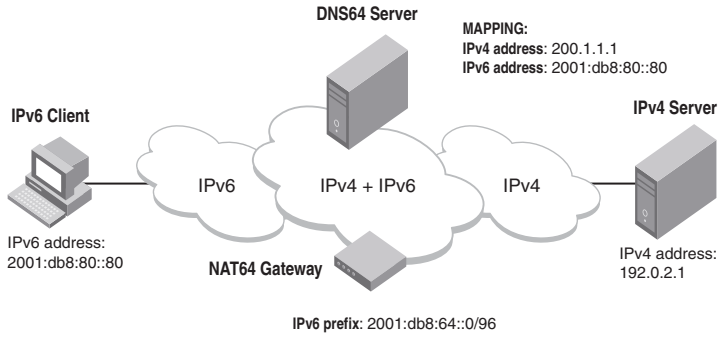
Stateless NAT64 overview

When configured as a stateless NAT64 gateway, ServerIron ADX enables IPv6-only clients to communicate with IPv4-only resources by means of address and packet translation. In a stateless NAT64 configuration, ServerIron ADX uses a mapping table to match IPv6 request packets sent from the IPv6 client to an IPv4 destination address of an IPv4 resource.

Stateless NAT64 components

Figure 7 shows a ServerIron ADX NAT64 configuration in which an IPv6-only client resides in an IPv6 network and an IPv4-only server (resource) resides in an IPv4 network. A DNS server and a ServerIron ADX communicate with both the IPv6 and IPv4 networks.

FIGURE 7 IPv6-only client to IPv4 resource



The DNS64 server provides the IPv6 client with a synthesized IPv6 address which enables the IPv6 client to reach the IPv4 resource. The synthesized IPv6 address consists of the NAT64 IPv6 prefix concatenated to the IPv4 destination address of the IPv4 resource and represents that IPv4 resource to the IPv6 network.

The ServerIron ADX is configured as a stateless NAT64 gateway. The NAT64 gateway strips the IPv6 prefix from the synthesized destination address and forwards the packet to the resource using an IPv4 address obtained from the IPv4-IPv6 mapping table as the source address.

A ServerIron ADX configured for stateless NAT64 translation requires the following components:

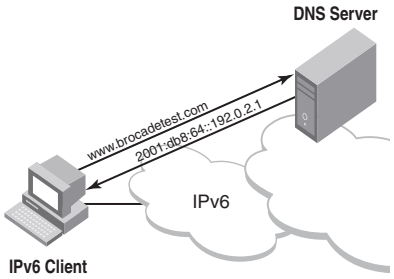
- DNS64 server: This component (not supplied by Brocade) is responsible for providing the IPv6 clients with the IPv6 addresses defined for IPv4 resources that the client wants to utilize. The DNS64 server provides the IPv6 client with a synthesized IPv6 address that enables the client to reach IPv4 resources. The synthesized IPv6 address consists of the NAT64 IPv6 prefix concatenated to the IPv4 destination address of the IPv4 resource and represents the IPv4 resource to the IPv6 network.
- NAT64 gateway: This component is installed on a ServerIron ADX. The NAT64 gateway is reached by the IPv6 client using an IPv6 address supplied to it by the DNS server. A mapping table translates IPv6 and IPv4 addresses. The mapping table can be populated statically or dynamically (by way of a real-time DNS query or prefetched DNS query).
- IPv6 prefix: An IPv6 prefix must be configured on the NAT64 gateway. The stateless NAT64 gateway concatenates the IPv6 prefix to the IPv4 source address of the IPv4 resource to create a synthesized IPv6 address that represents the IPv4 resource to the IPv6 network.
- IPv4-IPv6 mapping table: The mapping table defines a one-to-one relationship between an IPv4 address and an IPv6 address. This table can be defined manually or generated dynamically. Dynamically generated mapping tables can be created in real-time or prefetched from a DNS server. For dynamic mapping, you must configure a DNS server IP address on the ServerIron ADX.

Operation of stateless NAT64

The following steps describe the operation of stateless NAT64 translation for the environment shown in Figure 7.

1. In the example shown in Figure 8, the IPv6 client sends a query to the DNS64 server for the IPv6 address of the resource “www.brocadetest.com”.
2. The DNS64 server responds to the client’s query with a synthesized IPv6 address (2001:db8:64::192.0.2.1) that represents the IPv4 resource to the IPv6 network. Notice that this synthesized IPv6 address is composed of a user-configured IPv6 prefix (2001:db8:64::) and the IP address of the IPv4-only server (192.0.2.1).

FIGURE 8 IPv4 client to DNS64 server communication

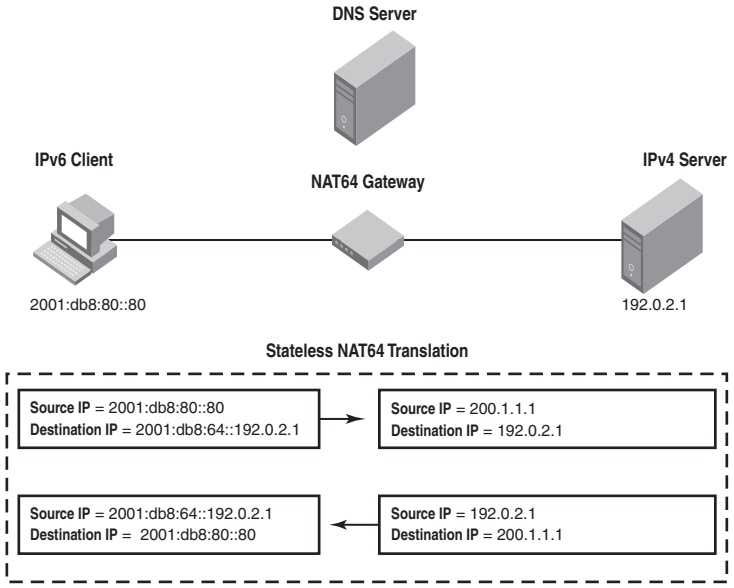


3. The request packet is then sent to the NAT64 gateway using the IPv6 address obtained from the DNS64 server as the destination IP (DIP) address (2001:db8:64::192.0.2.1.) and 2001:db8:80::80 as the source IP (SIP) address.
4. The NAT64 gateway strips away the initial portion of the IPv6 destination address leaving an IPv4 destination address (192.0.2.1) for the resource.
5. The NAT64 gateway uses its mapping table to translate the IPv6 SIP (2001:db8:80::80) into the IPv4 address that represents the IPv6 client to the IPv4 network (200.1.1.1).

   The mapping table used for this translation is populated statically or dynamically (by way of a real-time DNS query or prefetched DNS query). For more information, refer to "Populating the NAT64 mapping table" on page 30.
6. When the IPv4 resource sends a return packet to the IPv6 client, the NAT64 gateway uses the mapping table to translate the IPv4 DIP into the IPv6 address of the client: 2001:db8:80::80.
7. The NAT64 gateway concatenates the IPv6 prefix (2001:db8:64) to the IPv4 address of the resource (192.0.2.1) to create a synthesized IPv6 address that represents the IPv4 resource to the IPv6 network: 2001:db8:64::192.0.2.1.

The process occurs as shown in Figure 9.

FIGURE 9 Stateless NAT64 translation



Populating the NAT64 mapping table

The stateless NAT64 gateway uses a mapping table to translate an IPv6 client address to an IPv4 address.

This mapping table can be configured manually (using static mapping) or generated dynamically (using real-time dynamic learning or prefetched dynamic learning). In a stateless NAT64 configuration, all address mappings are one-to-one.

In configurations using static mapping, all entries in the NAT64 gateway mapping table are manually defined using the **nat64 map** command.

Dynamically generated mapping tables can be populated in real time by way of real-time dynamic learning or pre-populated by way of prefetched dynamic learning.

Stateless NAT64 static mapping configuration

In stateless NAT64 configurations using static mapping, the mapping table used to match IPv6 and IPv4 addresses is populated manually using the **nat64 map** command.

Route injection can be used in advanced static mapping configurations to inject IPv6 and IPv4 addresses into the respective network routing tables. Route distribution is then performed using any one of the routing protocols supported by ServerIron ADX: OSPF, IS-IS, and BGP (IPv6 to IPv4).

This section is divided into three topics:

- Basic stateless NAT64 static mapping configuration: This topic describes the tasks required to configure a basic stateless NAT64 gateway.
- Stateless NAT64 static mapping configuration with route injection: This topic describes the tasks required to configure a stateless NAT64 gateway that uses static route injection.

- Stateless NAT64 packet fragmentation configuration: This topic describes the options available for handling packet fragmentation in a stateless NAT64 gateway.

Basic stateless NAT64 static mapping configuration

A basic NAT64 gateway configuration uses a statically defined mapping table to make IPv4 resources available to IPv6 clients. The mapping table defines a one-to-one relationship between an IPv4 address and an IPv6 address. In a stateless NAT64 configuration that uses static mapping, all entries in the mapping table are configured manually.

The basic tasks required to configure a ServerIron ADX for basic NAT64 translation using static mapping of IPv6 and IPv4 addresses include the following:

- "Configuring stateless NAT64 IPv6 prefixes for static mappings" on page 31
- "Configuring stateless NAT64 static mappings" on page 31

Configuring stateless NAT64 IPv6 prefixes for static mappings

The NAT64 gateway uses a NAT64 IPv6 prefix to create a synthesized IPv6 address to represent IPv4 resources to the IPv6 network.

Use the **nat64 ipv6-prefix** command to specify the IPv6 prefix. In a stateless NAT64 configuration, you must include the **stateless** parameter. Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route**] **stateless**

The <*prefix/length*> variable specifies the IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT64 gateway configurations. Only one IPv6 prefix can be designated as stateless using this option.

In advanced static mapping configurations, the **inject-static-route** option can be used to advertise a route to mapped IPv4 addresses to the IPv6 network. For more information, refer to "Configuring static NAT64 IPv6 prefixes with route injection" on page 33.

Configuring stateless NAT64 static mappings

In static NAT64 configurations, use the **nat64 map** command to specify the IPv4 address of a resource and map that IPv4 address to the IPv6 address of a client. Enter a command such as the following:

```
ServerIron ADX(config)# nat64 map 201.1.1.100 2001:db8:8000::100
```

Syntax: [**no**] **nat64 map** <*ipv4-address*> <*ipv6-address*>

The <*ipv4-address*> variable specifies the IPv4 address for the IPv4-IPv6 mapping. The <*ipv6-address*> variable specifies an IPv6 address for the IPv4-IPv6 mapping.

NOTE
A maximum of 1024 entries is supported in the mapping table. Entries can be defined manually using the **nat64 map** command or dynamically using real-time or prefetched dynamic mappings. For more information, refer to "Populating the NAT64 mapping table" on page 30.

Stateless NAT64 static mapping with route injection

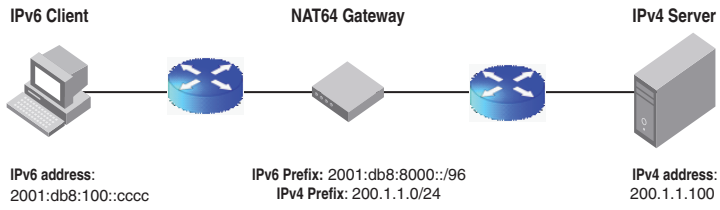
Route injection can be used to make the addresses assigned to IPv4 and IPv6 translations available as destinations in the routing tables of the respective IPv4 and IPv6 networks. Once these routes are injected into the respective routing tables, route distribution is then performed using any one of the routing protocols supported by ServerIron ADX: OSPF, IS-IS, and BGP (IPv6 to IPv4).

NAT64 route injection requires that the ServerIron ADX have a routing adjacency relationship with a router for the protocols with which you support route distribution.

- IPv4 route injection: For resources that are mapped to an IPv4 address, the IPv4 subnet is defined by the IPv4 prefix specified with the **nat64 ipv4-prefix** command. This command has the **inject-static-route** option which directs the ServerIron ADX to inject a static route to each of the IPv4 addresses within the defined IPv4 subnet that have an IPv6 address mapped to them. This mapping can be either performed using DNS dynamic mapping or though creation of a static entry using the **nat64 map** command. This destination is advertised to the adjacent neighbor and made available in routing tables of the IPv4 network.
- Starting with ServerIron ADX release 12.4.00, configuring the **inject-static-route** option with the **nat64 ipv6-prefix** command allows the configured IPv6 routing protocol to advertise the subnet defined by the IPv6 prefix to the adjacent IPv6 neighbor and thereby make it available in the routing tables of the IPv6 network, The next hop to the subnet is advertised to the IPv6 network as the adjacent router. In all prior releases of the ServerIron ADX, you are required specify an IPv6 interface on the ServerIron ADX (Ethernet or VE) that is directly connected to the adjacent router.

Figure 10 shows a typical IPv6-only client to IPv4 resource topology configured with router adjacency relationships on both the IPv6 and IPv4 sides of the NAT64 gateway. In this configuration, routes defined by the IPv6 prefix and IPv4 prefix are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured.

FIGURE 10 Stateless NAT64 route injection



To accomplish this redistribution, you must configure the routing protocol on your ServerIron ADX with the **redistribute static** command.

NOTE
For details about how to configure routing protocols on a ServerIron ADX, refer to the following chapters in the *ServerIron ADX Switch and Router Guide*: “Configuring OSPF”, “Configuring IPv6 Dynamic Routing”, “Configuring IS-IS (IPv4)”, “Configuring IPv6 IS-IS”, “Configuring BGP4 (IPv4)”, and “Configuring BGP4+”.

Tasks to configure a ServerIron ADX for advanced static mapping with route injection include the following:

- "Configuring static NAT64 IPv6 prefixes with route injection" on page 33
- "Configuring static NAT64 IPv4 prefixes with route injection" on page 33
- "Configuring NAT64 static mapping" on page 34

Configuring static NAT64 IPv6 prefixes with route injection

The NAT64 gateway uses a NAT64 IPv6 prefix to create a synthesized IPv6 address to represent IPv4 resources to the IPv6 network.

To specify an IPv6 prefix and a route to the subnet defined by that prefix, enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 inject-static-route
stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route**] **stateless**

The <*prefix/length*> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT64 gateway configurations. Only one IPv6 prefix can be designated as stateless using this option.

The **inject-static-route** option is used to advertise the subnet defined by the <*prefix/length*> variable on the IPv6 network.

In ServerIron ADX releases prior to 12.4.00, you must also identify either an Ethernet or VE interface and port number on the NAT64 gateway for static route injection. The specified interface must have an IPv6 address and be directly connected to an adjacent router.

Syntax: **[no] nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route** {**ve** <*port-number*> | **ethernet** <*port-number*>}]

The **ve** <*port-number*> or **ethernet** <*port-number*> variable specifies an IPv6 interface that is connected to the adjacent IPv6 router.

Configuring static NAT64 IPv4 prefixes with route injection

In a NAT64 configuration, the IPv4 prefix defines the range of IPv4 addresses that can be used to represent the IPv4 resources to the IPv6 network.

Use the **nat64 ipv4-prefix** command to specify an IPv4 prefix with a subnet mask. By including the **inject-static-route** option, you can direct the ServerIron ADX to inject a static route to every mapped address within the subnet defined by the IPv4 prefix.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv4-prefix 200.1.1.0/24 inject-static-route
```

Syntax: **[no] nat64 ipv4-prefix** <*prefix/subnet*> **[inject-static-route]**

The <*prefix/subnet*> variable specifies the NAT64 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway.

The **inject-static-route** option injects the host route into the routing protocol. The host route is only injected if the static map command is issued or a dynamic mapping is found. Unlike when an IPv4 prefix route is injected, the IPv6 route injection configuration does not require that you specify an interface. Route injection for IPv6 uses the null0 route.

Configuring NAT64 static mapping

Use the **nat64 map** command to specify the IPv4 address (within the subnet defined by the NAT64 IPv4 prefix) of a resource and map that address to the IPv6 address of a client.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 map 201.1.1.100 2001:db8:8000::100
```

Syntax: [**no**] **nat64 map** <*ipv4-address*> < *ipv6-address*>

The <*ipv4-address*> variable specifies the IPv4 address for the IPv4-IPv6 mapping. The <*ipv6-address*> variable specifies an IPv6 address for the IPv4-IPv6 mapping.

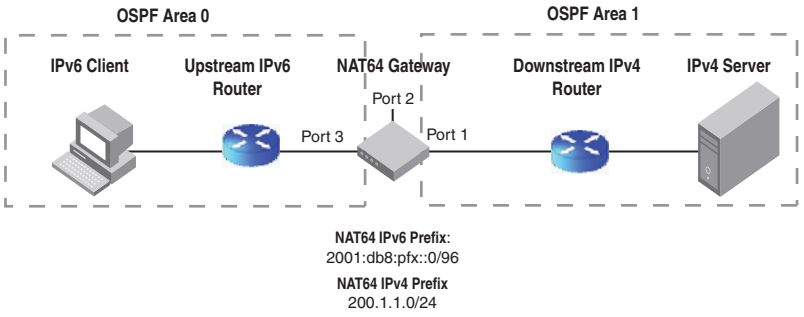
NOTE
The static IPv4 host route is injected into the IPv4 routing table if a matching IPv4 prefix is configured.

NOTE
A maximum of 1024 entries is supported in the mapping table. Entries can be defined manually using the **nat64 map** command or dynamically using real-time or prefetched dynamic mappings. For more information, refer to "Populating the NAT64 mapping table" on page 30.

Stateless NAT64 static route injection configuration example

Figure 11 shows a typical IPv6-only client to IPv4 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX (NAT64 gateway).

FIGURE 11 Stateless NAT64 route injection example



The following example configures a ServerIron ADX with NAT64 for route injection into OSPF with the following required details:

1. An IPv6 address is configured: VLAN 1 is configured with VE 7, which has an IPv6 address (2001:db8:80::80).

```
ServerIron ADX(config)# vlan 1
ServerIron ADX(config-vlan-1)# interface ethernet 1 to 2
```

```
ServerIron ADX(config-vlan-1)# interface ve 7
ServerIron ADX(config-vlan-1)# exit
ServerIron ADX(config)# interface ve 7
ServerIron ADX(config-vif-7)# ipv6 address 2001:db8:80::80
ServerIron ADX(config-vif-7)# exit
```

2. OSPF and OSPFv6 are configured for static route redistribution. The IPv6 side is configured as OSPF Area 0 and the IPv4 side is configured as OSPF Area 1.

```
ServerIron ADX(config)# ipv6 router ospf
ServerIron ADX(config-ospf6-router)# redistribute static
ServerIron ADX(config-ospf6-router)# area 0
ServerIron ADX(config-ospf6-router)# exit
ServerIron ADX(config)# router ospf
ServerIron ADX(config-ospf-router)# redistribute static
ServerIron ADX(config-ospf-router)# area 1
```

3. The NAT64 IPv4 prefix is configured with static route injection.

```
ServerIron ADX(config)# nat64 ipv4-prefix 200.1.1.0/24 inject-static-route
```

4. The NAT64 IPv6 prefix is configured with static route injection.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:64::0/96
inject-static-route stateless
```

   If you are running a ServerIron ADX build prior to 12.4.00, you must specify an interface and port number.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96
inject-static-route stateless ve 7 stateless
```

5. An IPv4 address (200.1.1.100) within the subnet defined by the NAT64 IPv4 prefix is mapped to the IPv6 address (2001:db8:80::80) specified in step 1.

```
ServerIron ADX(config)# nat64 map 200.1.1.100 2001:db8:80::80
```

In a static NAT64 configuration such as this one, the only routes injected into the local routing table are routes for IPv4 addresses that are mapped specifically to IPv6 addresses. In this example, only the route to 200.1.1.1 will be distributed to the IPv4 network and not the route to the entire subnet.

In dynamic NAT64 configurations, routes for all IPv4 addresses that are learned are distributed. For more information, refer to "Stateless NAT64 dynamic mapping configuration" on page 36.

NOTE
While this example describes NAT64 route injection for the OSPF protocol, you can also use this feature with BGP and IS-IS. Only the routing protocol configurations will differ.

Stateless NAT64 packet fragmentation configuration

Packets from the IPv6 client to the IPv4 server can be too large and need to be split into two IPv4 packets. The following describes the criteria for judging that packets are too large:

- Regular packets: IP packet total length greater than 1480 bytes
- Fragmented packets: IP packet total length greater than 1480 + 8 bytes

If the packets exceed these limitations, one of the following actions will be taken:

- If the **ipv6 frag-full-4to6** command is configured, the packet will be split and no further actions will be performed.
- If the condition in step 1 is not met, and the DF bit is set at the server, the “fragmentation needed” ICMP error message will be sent.
- If the conditions in steps 1 and 2 are not met, the packet will be split.

The **ipv6 frag-full-4to6** command is configured as shown in the following example.

```
ServerIronADX(config)# ipv6 frag-full-4to6
```

Syntax: [no] ipv6 frag-full-4to6

For more information about NAT64 fragmentation support, refer to “NAT64 fragmentation support” on page 5.

NOTE
The **ipv6-frag-full-4to6** command was introduced in ServerIron ADX release 12.4.00 and it replaces the **frag-664-reverse-full-sized-pkt** command from earlier releases.

Stateless NAT64 dynamic mapping configuration

The stateless NAT64 gateway uses a mapping table to translate the IPv6 addresses to IPv4 addresses and vice versa. This mapping table can be configured manually (using static mapping) or generated dynamically (using real-time dynamic learning or prefetched dynamic learning).

Dynamically generated mapping tables can be populated in real time by way of real-time dynamic learning or pre-populated by way of prefetched dynamic learning. You must configure a DNS server IP address on the ServerIron ADX to use dynamic mapping.

- Real-time dynamic learning: If a packet is received at the NAT64 gateway with an IPv6 destination address within the range defined by the NAT64 IPv6 prefix and it does not contain an entry in its mapping table for that IPv6 address, the gateway will send an IPv6 pointer (PTR) query to the DNS server to obtain the host name of the resource it is trying to reach. The NAT64 gateway then sends a query for the host name to determine the corresponding IPv4 address. The mapping defined by this operation is then entered into the mapping table of the NAT64 gateway.
- Prefetched dynamic learning: The NAT64 gateway can be configured to periodically send IPv6 PTR queries to the DNS server to identify IPv4 address translations for each of the IPv6 destination addresses defined within the IPv6 prefix subnet. The NAT64 gateway uses this information to populate its mapping table.

NAT64 real-time dynamic mapping configuration

The tasks required to configure a ServerIron ADX for NAT64 translation using mapping tables that are dynamically generated in real time include the following:

- “Configuring NAT64 IPv6 prefixes with real-time dynamic learning” on page 37
- “Configuring NAT64 IPv4 prefixes with real-time dynamic learning” on page 37
- “Configuring NAT64 with real-time dynamic learning” on page 37

Advanced configuration tasks include "Configuring NAT64 hold-off intervals for DNS discoveries" on page 38.

Configuring NAT64 IPv6 prefixes with real-time dynamic learning

The NAT64 gateway uses a NAT64 IPv6 prefix to create a synthesized IPv6 address to represent IPv4 resources to the IPv6 network.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/subnet*> **stateless**

The <*prefix/subnet*> variable specifies the IPv4 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway.

The **stateless** operand is required for stateless NAT64 gateway configurations. Only one IPv6 prefix can be designated as stateless using this option.

Configuring NAT64 IPv4 prefixes with real-time dynamic learning

In a NAT64 configuration, the IPv4 prefix defines the range of IPv4 addresses that can be used to represent the IPv6 clients to the IPv4 network.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv4-prefix 200.1.1.0/24
```

In this example, any IPv4 address within the subnet "200.1.1." can be assigned to an IPv6 client the NAT64 gateway makes available on the IPv4-only network.

Syntax: [**no**] **nat64 ipv4-prefix** <*prefix/subnet*>

The <*prefix/subnet*> variable specifies the NAT64 IPv4 prefix used by the ServerIron ADX when it operates as a stateless NAT64 gateway.

Configuring NAT64 with real-time dynamic learning

You can configure a ServerIron ADX to perform dynamic learning of IPv6 to IPv4 mappings through DNS as shown in the following example.

```
ServerIron ADX(config)# nat64 dns-dynamic-learning
```

Syntax: [no] nat64 dns-dynamic-learning

With this command configured, a ServerIron ADX will discover IPv4 to IPv6 mappings through DNS whenever the DNS server receives a new IPv6 destination or IPv4 source. With this command configured, you can configure a ServerIron ADX to prefetch mappings for IPv6 prefixes using the **nat64 ipv4-prefix** <*prefix/subnet*> **prefetch** command. For more information, refer to "Configuring NAT64 IPv4 prefixes with prefetched dynamic learning" on page 38.

You can clear all entries created using dynamic learning as described in "Clearing dynamic IPv4-IPv6 mappings" on page 41.

Configuring NAT64 hold-off intervals for DNS discoveries

By default, a DNS discovery (or refresh) fails if three retries time out or if the DNS server returns an error. In this situation, the NAT64 gateway can still receive traffic intended for IPv4 resources.

Use the **nat64 dns-fail-holdoff** command to direct the ServerIron ADX to wait a specified period of time (in seconds) before retrying a request to the DNS server. In the following example, the hold-off interval is set to 300 seconds:

```
ServerIron ADX(config)# nat64 dns-fail-holdoff 300
```

Syntax: [**no**] **nat64 dns-fail-holdoff** <*holdoff-interval*>

The <*holdoff-interval*> variable is configured in seconds. Configurable values are from 10 through 3600 seconds. The default value is 180 seconds.

NAT64 prefetched dynamic mapping configuration

The NAT64 gateway can be configured prefetched dynamic mappings—to periodically send PTR queries to the DNS server to determine the IPv4 address translations for the IPv6 destination addresses specified. The NAT64 gateway uses this information to populate its mapping table.

The tasks required to configure a ServerIron ADX for NAT64 translation using mapping tables that are dynamically generated using prefetched queries include the following:

- "Configuring NAT64 IPv6 prefixes with prefetched dynamic learning" on page 38
- "Configuring NAT64 IPv4 prefixes with prefetched dynamic learning" on page 38
- "Configuring NAT64 prefetched dynamic learning" on page 39

Advanced configuration tasks include "Configuring NAT64 hold-off intervals for DNS discoveries" on page 39.

Configuring NAT64 IPv6 prefixes with prefetched dynamic learning

The NAT64 gateway uses a NAT64 IPv6 prefix to create a synthesized IPv6 address to represent IPv4 resources to the IPv6 network.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/subnet*> **stateless**

The <*prefix/subnet*> variable specifies the IPv4 prefix that will be used by the ServerIron ADX when operating as a NAT64 gateway.

The **stateless** operand is required for stateless NAT64 gateway configurations. Only one IPv6 prefix can be designated as stateless using this option.

Configuring NAT64 IPv4 prefixes with prefetched dynamic learning

In a NAT64 configuration, the IPv4 prefix defines the range of IPv4 addresses that can be used to represent the IPv4 resources to the IPv6 network.

Use the **nat64 ipv4-prefix** command to specify an IPv4 prefix with a subnet mask. Any IPv4 address within the defined subnet can then be assigned to an IPv6 host made available through the NAT64 gateway.

To configure an IPv4 prefix for prefetched dynamic learning, enter a command such as the following using the **prefetch** option:

```
ServerIron ADX(config)# nat64 ipv4-prefix 200.1.1.0/24 prefetch
```

In this example, any IPv4 address within the subnet "2001:db8:8000::x" can be assigned to an IPv6 host on the NAT64 gateway thereby making it available to the IPv4-only network.

Syntax: [**no**] **nat64 ipv4-prefix** <*prefix/subnet*> [**prefetch**]

The <*prefix/subnet*> variable specifies the NAT64 IPv4 prefix used by the ServerIron ADX when it operates as a NAT64 gateway.

The **prefetch** option directs the ServerIron ADX to prefetch IPv6 to IPv4 mappings from DNS.

NOTE
The **nat64 dns-dynamic-learning** command must be configured for the **prefetch** option to take effect. If dynamic learning is not configured, an error message is displayed.

Configuring NAT64 prefetched dynamic learning

Use the **nat64 dns-dynamic-learning** command to configure a ServerIron ADX to perform dynamic learning of IPv6 to IPv4 mappings through DNS.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 dns-dynamic-learning
```

Syntax: [no] nat64 dns-dynamic-learning

With this command configured, a ServerIron ADX will discover IPv4 to IPv6 mappings through DNS whenever the DNS server receives a new IPv4 destination or IPv6 source.

Once this command is configured, you can configure a ServerIron ADX to prefetch mappings for IPv4 prefixes using the **nat64 ipv4-prefix** <*prefix/subnet*> **prefetch** command. For more information, refer to "Configuring NAT64 IPv4 prefixes with prefetched dynamic learning" on page 38.

Use the **clear nat64 dns-dynamic-mapping** command to clear all entries created using dynamic learning. For more information, refer to "Clearing dynamic IPv4-IPv6 mappings" on page 41.

Configuring NAT64 hold-off intervals for DNS discoveries

By default, a DNS discovery (or refresh) fails if three retries time out or if the DNS server returns an error. In this situation, the NAT64 gateway can still receive traffic intended for IPv4 resources.

Use the **nat64 dns-fail-holdoff** command to direct the ServerIron ADX to wait a specified period of time (in seconds) before retrying a request to the DNS server. In the following example, the hold-off interval is set to 300 seconds:

```
ServerIron ADX(config)# nat64 dns-fail-holdoff 300
```

Syntax: [**no**] **nat64 dns-fail-holdoff** <*holdoff-interval*>

The <*holdoff-interval*> variable is configured in seconds. Configurable values are from 10 through 3600 seconds. The default value is 180 seconds.

High availability for NAT64

The only high availability (HA) mode currently supported with the NAT64 feature is Active-Active HA. Support for this mode is essentially the same as for the IPv4 client to IPv4 resource configuration as described in "High availability for stateful NAT64" on page 18 with the following differences:

- Each ServerIron ADX is configured with a NAT64 IPv4 prefix, specifying the IPv4 address range, and a NAT64 IPv6 prefix, specifying the IPv6 address range.
- Because the NAT64 configuration does not use a port pool, the **port pool range** option is not configured.
- Because the NAT64 configuration is stateless, the **inject-active-only** option for HA configuration is not used.

Displaying NAT64 information

You can display the following information for a stateless NAT64 configuration:

- "Displaying NAT64 IPv6 to IPv4 address mappings" on page 40
- "Displaying in-progress dynamic NAT64 mappings" on page 41

NOTE
You can run the **show nat64 statistics** or the **show nat64 resources** commands to view information about stateless NAT64 configurations. For more information on these commands, refer to "Displaying NAT64 statistics" on page 21 or "Displaying NAT64 resources" on page 25.

Displaying NAT64 IPv6 to IPv4 address mappings

Use the **show nat64 map** command to display NAT64 IPv6 to IPv4 address mappings in stateless NAT64 gateways. This output is displayed as follows:

```
ServerIron ADX# show nat64 map all

******************************
NAT64 IPv6-IPv4 Address Mapping
IPv6 Address    IPv4 Address        Type
1.1.1.1         2001::1             CLI
1.1.1.2         2001::2             DNS
1.1.1.3                             DNS pending
******************************
```

Syntax: show nat64 map { <*IPv6_address*> | <*IPv4_address*> | **all** }

The <*IPv6_address*> variable specifies the IPv6 address for the IPv6-IPv4 mapping that you want to display.

The <*IPv4_address*> variable specifies the IPv4 address for the IPv6-IPv4 mapping that you want to display.

The **all** parameter displays all of the configured static NAT64 IPv6-IPv4 address mappings.

Table 7 describes the fields returned by the **show nat64 map** command.

TABLE 7 Display fields for show nat64 map all command

Field	Description
IPv6 Address	IPv6 address (destination for incoming IPv6 packets).
IPv4 Address	IPv4 address (source of incoming IPv4 packets).
Type	CLI: Configured DNS: Dynamically learned DNS pending: Dynamic learning ongoing

Displaying in-progress dynamic NAT64 mappings

Use the **show nat64 dns-in-flight** command to display in progress NAT64 DNS dynamic learning on stateless NAT64 gateways.

Syntax: show nat64 dns-in-flight

Table 8 describes the fields returned by the **show nat64 dns-in-flight** command.

TABLE 8 Display fields for show nat64 dns-in-flight command

Field	Description
Type	IPv6: Discovering IPv6 for known IPv4 address. IPv4: Discovering IPv4 for known IPv6 address.
Tries	The number of times this query was attempted.
DNS Server	DNS server used for this query.
Query	DNS query type sent; can be A, AAAA, or PTR.
IP Address	Known IP address (IPv6 or IPv4).

Clearing NAT64 information

You can clear the following information from a NAT64 configuration:

- Dynamic IPv4-IPv6 mappings
- NAT64 statistics

Clearing dynamic IPv4-IPv6 mappings

Use the **clear nat64 dns-dynamic-mapping** command to clear IPv4-IPv6 mappings learned through DNS on a ServerIron ADX. This command is issued as follows:

```
ServerIron ADX# clear nat64 dns-dynamic-mapping 1.1.1.1 2001:db8:7000
```

Syntax: clear nat64 dns-dynamic-mapping { <*IPv4_address*> | <*IPv6_address*> | **all** }

The <*IPv4_address*> variable specifies the IPv4 address for the IPv6-IPv4 mapping that you want to clear.

The <*IPv6_address*> variable specifies the IPv6 address for the IPv6-IPv4 mapping that you want to clear.

The **all** parameter clears all of the configured stateless static NAT64 IPv6-IPv4 address mappings.

Clearing stateless NAT64 statistics

Use the **clear nat64 statistics** command to clear stateless NAT64 statistics on a ServerIron ADX. This command is issued as follows:

```
ServerIron ADX# clear nat64 statistics
```

Syntax: clear nat64 statistics

Debugging stateless NAT64 configurations

Use the **debug nat64** command to enable diagnostic debugging of stateless NAT64 configurations.

To debug a stateless NAT64 configuration, you must use the **stateless** operand such as in the following example.

```
ServerIron ADX# debug nat64 stateless 192.0.2.1
```

In the example, diagnostic debugging is enabled for the IPv4 host at the IP address 192.0.2.1.

Syntax: debug nat64 stateless all | <*ipv4-address*> | <*ipv6-address*>

The **all** operand displays debug information for all IPv4 and IPv6 host IP addresses.

The <*ipv4-address*> variable displays debug information for a specific IPv4 host IP address.

The <*ipv6-address*> variable displays debug information for a specific IPv6 host IP address.

Use the **undebug** command to disable debugging.

Chapter

4

Stateless NAT46 Configuration

In this chapter

- Stateless NAT46 overview 43
- NAT46 static mapping configuration 45
- Stateless NAT46 dynamic mapping configuration 52
- High availability for NAT46 55
- Displaying NAT46 information 56
- Clearing NAT46 information 57
- Debugging NAT46 configurations 58

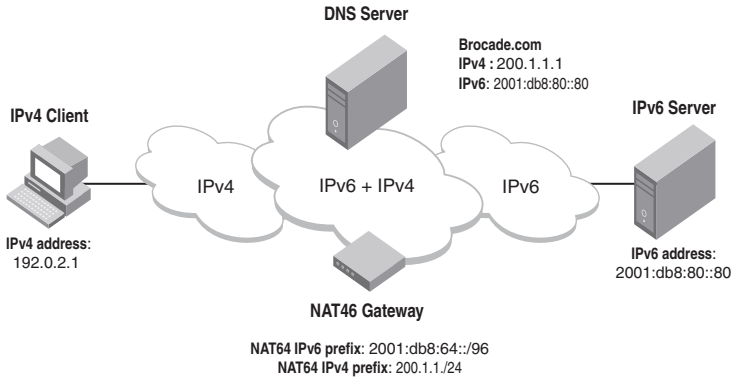
Stateless NAT46 overview

When configured as a NAT46 gateway, ServerIron ADX enables IPv4-only clients to communicate with IPv6 resources by way of address and packet translation. In a stateless NAT46 configuration, ServerIron ADX uses a mapping table to match IPv4 request packets sent from the IPv4 client to an IPv6 destination address of the IPv6 resource.

Stateless NAT46 components

Figure 12 shows a high-level view of a ServerIron ADX NAT46 configuration in which an IPv4-only client resides in an IPv4 network and an IPv6-only server (resource) resides in an IPv6 network. The DNS server and the ServerIron ADX communicate with both the IPv4 and IPv6 networks.

FIGURE 12 IPv4 client to IPv6 resource



The ServerIron ADX is configured as a stateless NAT46 gateway. For an IPv4 host, the NAT46 gateway generates the IPv6 source address by concatenating the IPv6 prefix to the IPv4 source address. The IPv6 destination address is obtained from the IPv4-IPv6 mapping table using the IPv4 destination address.

Two key components must be configured on the NAT46 gateway for it to enable communication between IPv4 clients and IPv6 resources.

- IPv6 prefix: The stateless NAT46 gateway is also configured with an IPv6 prefix that it concatenates to an IPv4 source address to create an IPv6 source address on the NAT46 gateway for each of the IPv4 clients to the IPv6 resources. The IPv6 prefix defined on this NAT46 gateway is 2001:db8:64::/96.
- IPv4-IPv6 mapping table: The mapping table defines a one-to-one relationship between an IPv4 address and an IPv6 address. This table can be defined manually or generated dynamically. Dynamically generated mapping tables can be created in real-time or prefetched.

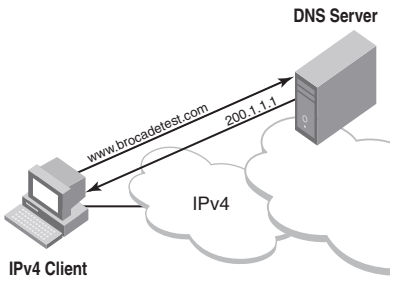
Operation of stateless NAT46

The following steps describe the operation of stateless NAT46 translation for the environment shown in Figure 12.

1. In the example shown in Figure 13, the IPv4 client sends a query to the DNS server for the IPv4 address of the resource “www.brocadetest.com” and the DNS server responds with the IPv4 address (200.1.1.1).

   The DNS server is configured to respond to a query from the IPv4 client with an IPv4 address.

FIGURE 13 IPv4 client to DNS server communication



2. The request packet is then sent to the NAT46 gateway using the IPv4 address (200.1.1.1) that was obtained from the DNS server for “brocadetest.com” as the destination IP (DIP) address.
3. The NAT46 gateway uses its mapping table to translate the IPv4 DIP (200.1.1.1) into the IPv6 address (2001:db8:80::80) of the IPv6 server that hosts “brocadetest.com”.

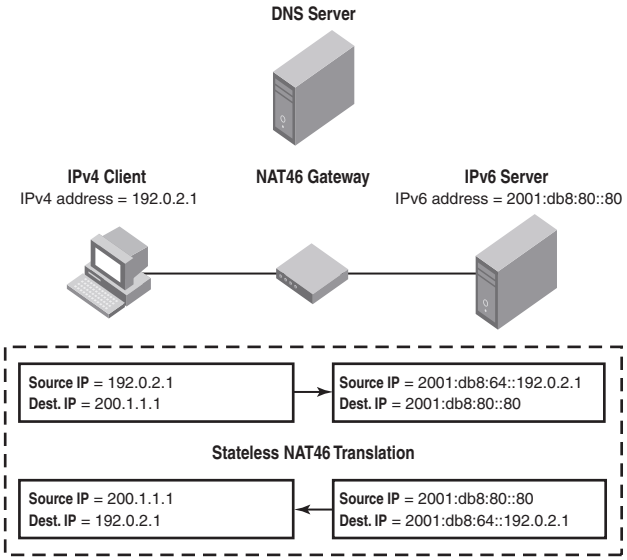
   The mapping table used for this translation can be populated statically or dynamically (by way of a real-time DNS query or prefetched DNS query). For more information, refer to “Populating the NAT46 mapping table” on page 45.
4. The NAT46 gateway creates a synthesized IPv6 source IP (SIP) address for the packet by concatenating the IPv6 prefix to the IPv4 address of the IPv6 resource.

   In this example, the IPv6 prefix (2001:db8:64::) is combined with the IPv4 address (192.0.2.1) to create the IPv6 source address (2001:db8:64::192.0.2.1) for the packet forwarded to the IPv6 resource.

5. The IPv6 client replies using the synthesized IP address (2001:db8:64::192.0.2.1) as the DIP and its own IPv6 address (2001:db8:80::80) as the SIP.
6. The NAT46 gateway strips out the IPv6 portion of the DIP (that is, the IPv6 prefix) and uses the remaining IPv4 portion (192.0.2.1) as the destination address.
7. The NAT46 gateway uses the mapping table to map the IPv6 SIP (2001:db8:80::80) to the IPv4 address that represents the resource to the IPv4 network (200.1.1.1). This is the same address that the DNS server uses for brocadetest.com.

The process occurs as shown in Figure 14.

FIGURE 14 Stateless NAT46 translation



Populating the NAT46 mapping table

The stateless NAT46 gateway uses a mapping table to translate the IPv4 destination address that the IPv4 network uses to identify the IPv6 resource into the actual IPv6 address of that resource.

This mapping table can be configured manually (using static mapping) or generated dynamically (using real-time dynamic learning or prefetched dynamic learning). A maximum of 1024 entries is supported in the mapping table.

In configurations using static mapping, all entries in the NAT46 gateway mapping table are manually defined using the **nat64 map** command.

Dynamically generated mapping tables can be populated in real time by way of real-time dynamic learning or pre-populated by way of prefetched dynamic learning. You must configure a DNS server IP address on the ServerIron ADX to use dynamic mapping.

NAT46 static mapping configuration

In NAT46 configurations using static mapping, the mapping table used to match IPv4 and IPv6 addresses is populated manually using the **nat64 map** command.

Route injection can be used in advanced static mapping configurations to inject IPv4 and IPv6 addresses into the respective network routing tables. Route distribution is then performed using any one of the routing protocols supported by ServerIron ADX: OSPF, IS-IS, and BGP (IPv4 to IPv6).

This section is divided into three topics:

- Basic NAT46 static mapping configuration: This topic describes the tasks required to configure a basic stateless NAT46 gateway.
- Stateless NAT46 static mapping configuration with route injection: This topic describes the tasks required to configure a stateless NAT46 gateway that uses static route injection.
- Stateless NAT46 packet fragmentation configuration: This topic describes the options available for handling packet fragmentation in a stateless NAT46 gateway.

Basic NAT46 static mapping configuration

A basic NAT46 gateway configuration uses a statically defined mapping table to make IPv6 resources available to IPv4 clients. The mapping table defines a one-to-one relationship between an IPv4 address and an IPv6 address. In a stateless NAT46 configuration that uses static mapping, all entries in the mapping table are configured manually.

Basic NAT46 static mapping configuration tasks include the following:

- "Configuring basic static NAT46 IPv6 prefixes" on page 46
- "Configuring basic NAT46 static mapping" on page 46

Configuring basic static NAT46 IPv6 prefixes

The NAT46 gateway uses a NAT46 IPv6 prefix to create a synthesized IPv6 source address on the NAT46 gateway for each of the IPv4 clients. The synthesized IPv6 address is used to represent the IPv4 clients to the IPv6 network.

Use the **nat64 ipv6-prefix** command to specify the IPv6 prefix. Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> **stateless**

The <*prefix/length*> variable specifies the NAT46 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT46 gateway configurations. Only one IPv6 prefix can be configured with this option.

Configuring basic NAT46 static mapping

In static NAT46 configurations, use the **nat64 map** command to specify an IPv4 address and map that IPv4 address to the IPv6 address of a resource. Enter a command such as the following:

```
ServerIron ADX(config)# nat64 map 201.1.1.100 2001:db8:8000::100
```

Syntax: [**no**] **nat64 map** < *ipv4-address*> < *ipv6-address*>

The <*ipv4-address*> variable specifies an IPv4 address for the IPv4-IPv6 mapping. The <*ipv6-address*> variable specifies an IPv6 address for the IPv4-IPv6 mapping.

NOTE
A maximum of 1024 entries is supported in the mapping table. Entries can be defined manually using the **nat64 map** command or dynamically using real-time or prefetched dynamic mappings. For more information, refer to "Populating the NAT46 mapping table" on page 45.

Stateless NAT46 static mapping with route injection

In NAT46 configurations using static mapping of IPv4 and IPv6 addresses, an IPv4 prefix is defined for an IPv4 subnet of mapped IPv4 addresses and an IPv6 prefix for an IPv6 subnet is used to represent the IPv4 clients to the IPv6 resources.

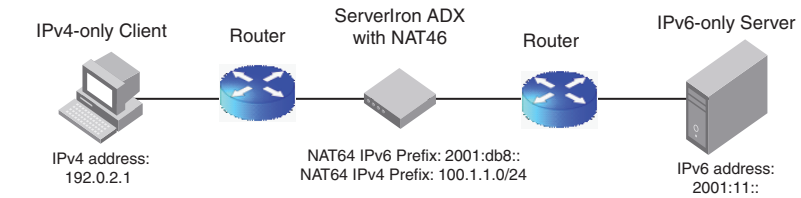
Route injection can be used to make the addresses assigned to these translations available as destinations in the routing tables of the respective IPv6 and IPv4 networks. Once these routes are injected into the respective routing tables, route distribution is then performed using any one of the routing protocols supported by ServerIron ADX: OSPF, IS-IS, and BGP (IPv4 to IPv6).

NAT46 route injection requires that the ServerIron ADX have a routing adjacency relationship with a router for the protocols for which you want route distribution.

- IPv4 route injection: For resources that are mapped to an IPv4 address, the IPv4 subnet is defined by the IPv4 prefix specified with the **nat64 ipv4-prefix** command. This command has the **inject-static-route** option which directs the ServerIron ADX to inject a static route to each of the IPv4 addresses within the defined IPv4 subnet that have an IPv6 address mapped to them. This mapping can be either performed using DNS dynamic mapping or though creation of a static entry using the **nat64 map** command. This destination is advertised to the adjacent neighbor and made available in routing tables of the IPv4 network.
- Starting with ServerIron ADX release 12.4.00, configuring the **inject-static-route** option with the **nat64 ipv6-prefix** command allows the configured IPv6 routing protocol to advertise the subnet defined by the IPv6 prefix to the adjacent IPv6 neighbor and thereby make it available in the routing tables of the IPv6 network, The next hop to the subnet is advertised to the IPv6 network as the adjacent router. In all prior releases of the ServerIron ADX, you are required specify an IPv6 interface on the ServerIron ADX (Ethernet or VE) that is directly connected to the adjacent router.

Figure 15 shows a typical IPv4-only client to IPv6 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX. In this configuration, routes defined by the IPv4 prefix and IPv6 prefix are advertised to the adjacent routers and distributed to the respective networks using the routing protocol configured.

FIGURE 15 NAT46 route injection



To accomplish this redistribution, you must configure the routing protocol on your ServerIron ADX with the **redistribute static** command.

NOTE
For details about how to configure routing protocols on a ServerIron ADX, refer to the following chapters in the *ServerIron ADX Switch and Router Guide*: "Configuring OSPF", "Configuring IPv6 Dynamic Routing", "Configuring IS-IS (IPv4)", "Configuring IPv6 IS-IS", "Configuring BGP4 (IPv4)", and "Configuring BGP4+".

Advanced NAT46 static mapping configuration tasks include the following:

- "Configuring static NAT46 IPv6 prefixes with static route injection" on page 48
- "Configuring static NAT46 IPv4 prefixes with static route injection" on page 49
- "Configuring NAT46 static mapping" on page 49

Configuring static NAT46 IPv6 prefixes with static route injection

The NAT46 gateway uses a NAT46 IPv6 prefix to create a synthesized IPv6 source address on the NAT46 gateway for each of the IPv4 clients. The synthesized IPv6 address is used to represent the IPv4 clients to the IPv6 network.

To specify an IPv6 prefix and a route to the subnet defined by that prefix, enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::/96 inject-static-route
stateless
```

In the example, an IPv6 prefix (2001:db8:8000::) is defined that specifies the IPv6 subnet. The **inject-static-route** option specifies a route to this subnet that will be injected into the local routing table.

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route**] **stateless**

The <*prefix/length*> variable specifies the NAT46 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT46 gateway configurations. Only one IPv6 prefix can be configured with this option.

The **inject-static-route** option is used to advertise the subnet defined by the <*prefix/length*> variable on the IPv6 network.

In ServerIron ADX releases prior to 12.4.00, you must also identify either an Ethernet or VE interface and port number on the NAT64 gateway for static route injection. The specified interface must have an IPv6 address and be directly connected to an adjacent router.

Syntax: **[no] nat64 ipv6-prefix** <*prefix/length*> [**inject-static-route** {**ve** <*port-number*> | **ethernet** <*port-number*>}]

The **ve** <*port-number*> or **ethernet** <*port-number*> variable specifies an IPv6 interface that is connected to the adjacent IPv6 router.

Configuring static NAT46 IPv4 prefixes with static route injection

The NAT46 IPv4 prefix provides a range of IPv4 addresses on the NAT46 gateway that can be used to represent IPv6 resources. It is created by defining an IPv4 prefix with a subnet mask. Any IPv4 address within the defined subnet can then be assigned to an IPv6 resource made available through the gateway.

Use the **nat64 ipv4-prefix** command with the **inject-static-route** option to direct the ServerIron ADX to inject a static route to each of the IPv4 addresses within the defined IPv4 subnet that have IPv6 addresses mapped to them.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv4-prefix 200.1.1.0/24 inject-static-route
```

Syntax: [**no**] **nat64 ipv4-prefix** <*prefix/subnet*> [**inject-static-route**]

The <*prefix/subnet*> variable specifies the NAT46 IPv4 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway.

The **inject-static-route** option injects the host route into the routing protocol. The host route is only injected if the static map command is issued or a dynamic mapping is found. Unlike when an IPv6 prefix route is injected, the IPv4 route injection configuration does not require that you specify an interface. Route injection for IPv4 uses the null0 route.

Configuring NAT46 static mapping

Use the **nat64 map** command to specify an IPv4 address (within the subnet defined by the NAT46 IPv4 prefix) and map that address to the IPv6 address of a resource.

Enter a command such as the following:

```
ServerIron ADX(config)# nat64 map 201.1.1.100 2001:db8:8000::100
```

Syntax: [**no**] **nat64 map** <*ipv4-address*> <*ipv6-address*>

The <*ipv4-address*> variable defines an IPv4 address within the subnet defined by the NAT46 IPv4 prefix that identifies an IPv6 resource to the IPv4-only network.

The <*ipv6-address*> variable specifies the IPv6 address of the IPv6 resource that is being mapped to the IPv4 address specified by the <*ipv4-address*> variable within this command.

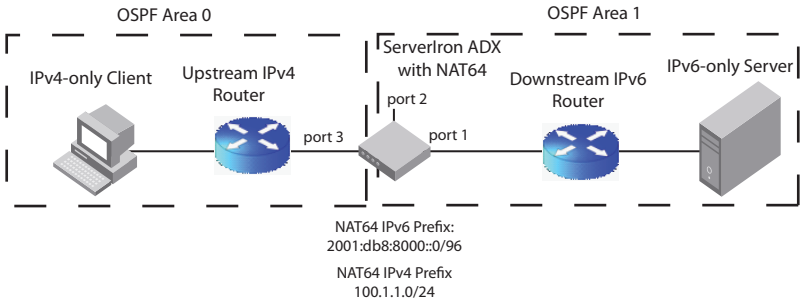
NOTE
The static IPv4 host route is injected into the IPv4 routing table if a matching IPv4 prefix is configured.

NOTE
A maximum of 1024 entries is supported in the mapping table. Entries can be defined manually using the **nat64 map** command or dynamically using real-time or prefetched dynamic mappings. For more information, refer to "Populating the NAT46 mapping table" on page 45.

Stateless NAT46 static route injection configuration example

Figure 16 shows a typical IPv4-only client to IPv6 resource topology configured with router adjacency relationships on both the IPv4 and IPv6 sides of the ServerIron ADX (NAT46 gateway).

FIGURE 16 NAT46 route injection example



The following configuration example shows the steps required to configure a stateless NAT46 gateway for route injection into OSPF.

1. An IPv6 address is configured: VLAN 1 is configured with VE 7, which has an IPv6 address (2001:db8:8000::1).

```
ServerIron ADX(config)# vlan 1
ServerIron ADX(config-vlan-1)# interface ethernet 1 to 2
ServerIron ADX(config-vlan-1)# interface ve 7
ServerIron ADX(config-vlan-1)# exit
ServerIron ADX(config)# interface ve 7
ServerIron ADX(config-vif-7)# ipv6 address 2001:db8:8000::1
ServerIron ADX(config-vif-7)# exit
```

2. OSPF and OSPFv6 are configured for static route redistribution. The IPv4 side is configured as OSPF Area 0 and the IPv6 side is configured as OSPF Area 1.

```
ServerIron ADX(config)# router ospf
ServerIron ADX(config-ospf-router)# redistribute static
ServerIron ADX(config-ospf-router)# area 0
ServerIron ADX(config-ospf-router)# exit
ServerIron ADX(config)# ipv6 router ospf
ServerIron ADX(config-ospf6-router)# redistribute static
ServerIron ADX(config-ospf6-router)# area 1
ServerIron ADX(config-ospf6-router)# exit
```

3. The NAT46 IPv4 prefix is configured with static route injection.

```
ServerIron ADX(config)# nat64 ipv4-prefix 100.1.1.0/24 inject-static-route
```

4. The NAT46 IPv6 prefix is configured with static route injection.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96
inject-static-route stateless
```

If you are running a ServerIron ADX build prior to 12.4.00, you must specify an interface and port number.

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96
inject-static-route stateless ve 7 stateless
```

5. An IPv4 address (100.1.1.100) within the subnet defined by the NAT46 IPv4 prefix is mapped to the IPv6 address (2001:db8:8000::100) specified in Step 1.

```
ServerIron ADX(config)# nat64 map 100.1.1.100 2001:db8:8000::100
```

In a static NAT46 configuration such as this one, the only routes injected into the local routing table are routes for IPv4 addresses that specifically are mapped to IPv6 addresses. In this example, only the route to 100.1.1.100 will be distributed to the IPv4 network and not the route to the entire subnet.

In dynamic NAT46 configurations, routes for all IPv4 addresses that are learned are distributed. For more information, refer to "Stateless NAT46 dynamic mapping configuration" on page 52.

NOTE
While this example describes NAT46 route injection for the OSPF protocol, you can also use this feature with BGP and IS-IS. Only the routing protocol configurations will differ.

Configuring NAT46 packet fragmentation

Packets from the IPv4 client to the IPv6 server can be too large and need to be split into two IPv6 packets. The following describes the criteria for judging that packets are too large:

- Regular packets: IP packet total length greater than 1480 bytes
- Fragmented packets: IP packet total length greater than 1480 + 8 bytes

If the packets exceed these limitations, one of the following actions will be taken.

- If the **ipv6 frag-full-4to6** command is configured, the packet will be split and no further actions will be performed.
- If the condition in step 1 is not met, and the DF bit is set at the server, the "fragmentation needed" ICMP error message will be sent.
- If the conditions in steps 1 and 2 are not met, the packet will be split.

The **ipv6 frag-full-4to6** command is configured as shown in the following example.

```
ServerIronADX(config)# ipv6 frag-full-4to6
```

Syntax: [no] ipv6 frag-full-4to6

For more information about fragmentation support, refer to "NAT64 fragmentation support" on page 5.

NOTE
The **ipv6-frag-full-4to6** command was introduced in ServerIron ADX release 12.4.00 and it replaces the **frag-664-reverse-full-sized-pkt** command from earlier releases.

Stateless NAT46 dynamic mapping configuration

The stateless NAT46 gateway uses a mapping table to translate the IPv4 destination address that the IPv4 network uses to identify the IPv6 resource into the actual IPv6 address of that resource.

This mapping table can be configured manually (using static mapping) or generated dynamically (using real-time dynamic learning or prefetched dynamic learning). A maximum of 1024 entries is supported in the mapping table.

In configurations using static mapping, all entries in the NAT46 gateway mapping table are manually defined using the **nat64 map** command.

Dynamically generated mapping tables can be populated in real time by way of real-time dynamic learning or pre-populated by way of prefetched dynamic learning. You must configure a DNS server IP address on the ServerIron ADX to use dynamic mapping.

- Real-time dynamic learning: If a packet is received at the NAT46 gateway with an IPv4 destination address within the range defined by the NAT46 IPv4 prefix and it does not contain an entry in its mapping table for that IPv4 address, the gateway will send an IPv4 PTR query to the DNS server to obtain the host name of the resource it is trying to reach. The NAT46 gateway then sends a query for the host name to determine the corresponding IPv6 address. The mapping defined by this operation is then entered into the mapping table of the NAT46 gateway.
- Prefetched dynamic learning: The NAT46 gateway can be configured to periodically send IPv4 PTR queries to the DNS server to identify IPv6 address translations for each of the IPv4 destination addresses defined within the IPv4 prefix subnet. The NAT46gateway uses this information to populate its mapping table.

Real-time NAT46 dynamic mapping configuration

Basic real-time NAT46 dynamic mapping configuration tasks include the following:

- "Configuring NAT46 IPv6 prefixes with real-time dynamic learning" on page 52
- "Configuring NAT46 IPv4 prefixes with real-time dynamic learning" on page 53
- "Configuring NAT46 real-time dynamic learning" on page 53

Advanced real-time NAT46 dynamic mapping configuration tasks include "Configuring NAT46 hold-off intervals for DNS discoveries" on page 53.

Configuring NAT46 IPv6 prefixes with real-time dynamic learning

The NAT46 gateway uses a NAT46 IPv6 prefix to create a synthesized IPv6 source address on the NAT46 gateway for each of the IPv4 clients. The synthesized IPv6 address is used to represent the IPv4 clients to the IPv6 network.

Use the **nat64 ipv6-prefix** command to specify the IPv6 prefix. Enter a command such as the following:

```
ServerIron ADX(config)# nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> **stateless**

The <*prefix/length*> variable specifies the NAT46 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT46 gateway configurations. Only one IPv6 prefix can be configured with this option.

Configuring NAT46 IPv4 prefixes with real-time dynamic learning

In a NAT46 configuration, the IPv4 prefix defines the range of IPv4 addresses that can be used to represent the IPv6 resources available to IPv4 clients.

Use the **nat64 ipv4-prefix** command to specify an IPv4 prefix with a subnet mask. Any IPv4 address within the defined subnet can then be assigned to an IPv6 resource made available through the NAT46 gateway.

Enter a command such as the following:

```
ServerIron ADX(config)#  nat64 ipv4-prefix 200.1.1.0/24
```

In this example, any IPv4 address within the subnet “200.1.1.x” can be assigned to an IPv6 resource in the IPv6-only network.

Syntax: [**no**] **nat64 ipv4-prefix** <*prefix/subnet*>

The <*prefix/subnet*> variable specifies the NAT46 IPv4 prefix used by the ServerIron ADX when it operates as a NAT46 gateway.

Configuring NAT46 real-time dynamic learning

You can configure a ServerIron ADX to perform dynamic learning of IPv4 to IPv6 mappings through DNS as shown in the following example.

```
ServerIron ADX(config)#  nat64 dns-dynamic-learning
```

Syntax: [no] nat64 dns-dynamic-learning

With this command configured, a ServerIron ADX will discover IPv6 to IPv4 mappings through DNS whenever the DNS server receives a new IPv4 destination or IPv6 source. With this command configured, you can configure a ServerIron ADX to prefetch mappings for IPv4 prefixes using the **nat64 ipv4-prefix** <*prefix/subnet*> **prefetch** command. For more information, refer to “Configuring NAT46 IPv4 prefixes with prefetched dynamic learning” on page 54.

You can clear all entries created using dynamic learning as described in “Clearing dynamic IPv6-IPv4 mappings” on page 57.

Configuring NAT46 hold-off intervals for DNS discoveries

In advanced configurations, you can configure hold-off intervals for DNS discoveries. By default, a DNS discovery (or refresh) fails if three retries time out or if the DNS64 server returns an error. In this situation, the NAT46 gateway can still receive traffic intended for IPv6 resources.

Use the **nat64 dns-fail-holdoff** command to direct the ServerIron ADX to wait a specified period of time (in seconds) before retrying a request to the DNS64 server. In the following example, the hold-off interval is set to 300 seconds:

```
ServerIron ADX(config)#  nat64 dns-fail-holdoff 300
```

Syntax: [**no**] **nat64 dns-fail-holdoff** <*holdoff-interval*>

The <*holdoff-interval*> variable is configured in seconds. Configurable values are from 10 through 3600 seconds. The default value is 180 seconds.

Prefetched NAT46 dynamic mapping configuration

The NAT46 gateway can be configured to prefetch dynamic mappings—to periodically send PTR queries to the DNS server to determine the IPv6 address translations for the IPv4 destination addresses specified. The NAT46 gateway uses this information to populate its mapping table.

Basic prefetched NAT46 dynamic mapping configuration tasks include the following:

- "Configuring NAT46 IPv6 prefixes with prefetched dynamic learning" on page 54
- "Configuring NAT46 IPv4 prefixes with prefetched dynamic learning" on page 54
- "Configuring NAT46 prefetched dynamic learning" on page 55

Advanced real-time NAT46 dynamic mapping configuration tasks include "Configuring NAT46 hold-off intervals for DNS discoveries" on page 55.

Configuring NAT46 IPv6 prefixes with prefetched dynamic learning

The NAT46 gateway uses a NAT46 IPv6 prefix to create a synthesized IPv6 source address on the NAT46 gateway for each of the IPv4 clients. The synthesized IPv6 address is used to represent the IPv4 clients to the IPv6 network.

Use the **nat64 ipv6-prefix** command to specify the IPv6 prefix. Enter a command such as the following:

```
ServerIron ADX(config)#  nat64 ipv6-prefix 2001:db8:8000::0/96 stateless
```

Syntax: [**no**] **nat64 ipv6-prefix** <*prefix/length*> **stateless**

The <*prefix/length*> variable specifies the NAT46 IPv6 prefix that will be used by the ServerIron ADX when operating as a NAT46 gateway. Currently, the only supported prefix length is 96.

The **stateless** operand is required for stateless NAT46 gateway configurations. Only one IPv6 prefix can be configured with this option.

Configuring NAT46 IPv4 prefixes with prefetched dynamic learning

In a NAT46 configuration, the IPv4 prefix defines the range of IPv4 addresses that can be used to represent the IPv6 resources available to IPv4 clients.

Use the **nat64 ipv4-prefix** command to specify an IPv4 prefix with a subnet mask. Any IPv4 address within the defined subnet can then be assigned to an IPv6 resource made available through the NAT46 gateway.

Use the **prefetch** option to enable that the NAT46 gateway to pre-populate the mapping table such as in the following command.

```
ServerIron ADX(config)#  nat64 ipv4-prefix 200.1.1.0/24 prefetch
```

In this example, any IPv4 address within the subnet "200.1.1.x" can be assigned to an IPv6 resource in the IPv6-only network.

Syntax: [**no**] **nat64 ipv4-prefix** <*prefix/subnet*>[**prefetch**]

The <*prefix/subnet*> variable specifies the NAT46 IPv4 prefix used by the ServerIron ADX when it operates as a NAT46 gateway.

The **prefetch** option directs the ServerIron ADX to prefetch IPv4 to IPv6 mappings from DNS.

NOTE
The **nat64 dns-dynamic-learning** command must be configured for the **prefetch** option to take effect. If dynamic learning is not configured, an error message is displayed.

Configuring NAT46 prefetched dynamic learning

Use the **nat64 dns-dynamic-learning** command to configure a ServerIron ADX to perform dynamic learning of IPv4 to IPv6 mappings through DNS.

Enter a command such as the following:

```
ServerIron ADX(config)#  nat64 dns-dynamic-learning
```

Syntax: [no] nat64 dns-dynamic-learning

With this command configured, a ServerIron ADX will discover IPv6 to IPv4 mappings through DNS whenever the DNS server receives a new IPv4 destination or IPv6 source.

Once this command is configured, you can configure a ServerIron ADX to prefetch mappings for IPv4 prefixes using the **nat64 ipv4-prefix** <*prefix/subnet*> **prefetch** command. For more information, refer to "Configuring NAT46 IPv4 prefixes with prefetched dynamic learning" on page 54.

Use the **clear nat64 dns-dynamic-mapping** command to clear all entries created using dynamic learning. For more information, refer to "Clearing dynamic IPv6-IPv4 mappings" on page 57.

Configuring NAT46 hold-off intervals for DNS discoveries

In advanced configurations, you can configure hold-off intervals for DNS discoveries. By default, a DNS discovery (or refresh) fails if three retries time out or if the DNS64 server returns an error. In this situation, the NAT46 gateway can still receive traffic intended for IPv6 resources.

Use the **nat64 dns-fail-holdoff** command to direct the ServerIron ADX to wait a specified period of time (in seconds) before retrying a request to the DNS64 server. In the following example, the hold-off interval is set to 300 seconds:

```
ServerIron ADX(config)#  nat64 dns-fail-holdoff 300
```

Syntax: [**no**] **nat64 dns-fail-holdoff** <*holdoff-interval*>

The <*holdoff-interval*> variable is configured in seconds. Configurable values from 10 through 3600 seconds. The default value is 180 seconds.

High availability for NAT46

The only high availability (HA) mode currently supported with the NAT46 feature is Active-Active HA. Support for this mode is essentially the same as for the IPv6 client to IPv6 resource configuration as described in "High availability for NAT64" on page 15 with the following differences:

- Each ServerIron ADX is configured with a NAT46 IPv6 prefix, specifying the IPv6 address range, and a NAT46 IPv4 prefix, specifying the IPv4 address range.
- Because the NAT46 configuration does not use a port pool, the **port pool range** option is not configured.
- Because the NAT46 configuration is stateless, the **inject-active-only** option for HA configuration is not used.

Displaying NAT46 information

You can display the following information for a NAT46 configuration:

- "Displaying NAT46 IPv4 to IPv6 address mappings" on page 56
- "Displaying in-progress dynamic NAT46 mappings" on page 57

NOTE
You can run the **show nat64 statistics** or the **show nat64 resources** commands to view information about stateless NAT46 configurations. For more information on these commands, refer to "Displaying NAT64 statistics" on page 21 or "Displaying NAT64 resources" on page 25.

Displaying NAT46 IPv4 to IPv6 address mappings

Use the **show nat64 map** command to display NAT46 IPv4 to IPv6 address mappings in stateless NAT46 gateways. This output is displayed as follows:

```
ServerIron ADX# show nat64 map all

*****************************
NAT64 IPv4-IPv6 Address Mapping
IPv4 Address    IPv6 Address        Type
1.1.1.1         2001::1             CLI
1.1.1.2         2001::2             DNS
1.1.1.3                             DNS pending
*****************************
```

Syntax: show nat64 map { <*IPv4_address*> | <*IPv6_address*> | **all** }

The <*IPv4_address*> variable specifies the IPv4 address for the IPv4-IPv6 mapping that you want to display.

The <*IPv6_address*> variable specifies the IPv6 address for the IPv4-IPv6 mapping that you want to display.

The **all** parameter displays all of the configured static NAT46 IPv4-IPv6 address mappings.

Table 9 describes the fields returned by the **show nat64 map all** command.

TABLE 9 Display fields for show nat64 map all command

Field	Description
IPv4 Address	IPv4 address (destination for incoming IPv4 packets).

TABLE 9 Display fields for show nat64 map all command

Field	Description
IPv6 Address	IPv6 address (source of incoming IPv6 packets).
Type	**CLI**: Configured **DNS**: Dynamically learned **DNS pending**: Dynamic learning ongoing

Displaying in-progress dynamic NAT46 mappings

Use the **show nat64 dns-in-flight** command to display in progress NAT46 DNS dynamic learning on stateless NAT46 gateways.

Syntax: show nat64 dns-in-flight

Table 10 describes the fields returned by the **show nat64 dns-in-flight** command.

TABLE 10 Display fields for show nat64 dns-in-flight command

Field	Description
Type	**IPv4**: Discovering IPv4 for known IPv6 address. **IPv6**: Discovering IPv6 for known IPv4 address.
Tries	The number of times this query was attempted.
DNS Server	DNS server used for this query.
Query	DNS query type sent; can be A, AAAA, or PTR.
IP Address	Known IP address (IPv4 or IPv6).

Clearing NAT46 information

You can clear the following information from a NAT46 configuration:

- Dynamic IPv6-IPv4 mappings
- NAT46 statistics

Clearing dynamic IPv6-IPv4 mappings

Use the **clear nat64 dns-dynamic-mapping** command to clear IPv6-IPv4 mappings learned through DNS on a ServerIron ADX. This command is issued as follows:

```
ServerIron ADX# clear nat64 dns-dynamic-mapping 1.1.1.1 2001:db8:7000
```

Syntax: clear nat64 dns-dynamic-mapping { <*IPv4_address*> | <*IPv6_address*> | **all** }

The <*IPv4_address*> variable specifies the IPv4 address for the IPv4-IPv6 mapping that you want to clear.

The <*IPv6_address*> variable specifies the IPv6 address for the IPv4-IPv6 mapping that you want to clear.

The **all** parameter clears all of the configured stateless static NAT46 IPv4-IPv6 address mappings.

Clearing stateless NAT46 statistics

Use the **clear nat64 statistics** command to clear stateless NAT46 statistics on a ServerIron ADX. This command is issued as follows:

```
ServerIron ADX# clear nat64 statistics
```

Syntax: clear nat64 statistics

Debugging NAT46 configurations

Use the **debug nat64** command to enable diagnostic debugging of NAT46 configurations.

To debug a NAT46 configuration, you must use the **stateless** operand such as in the following example:

```
ServerIron ADX# debug nat64 stateless 192.0.2.1
```

In the example, diagnostic debugging is enabled for the IPv4 host at the IP address 192.0.2.1.

Syntax: debug nat64 stateless all | <*ipv4-address*> | <*ipv6-address*>

The **all** operand displays debug information for all IPv4 and IPv6 host IP addresses.

The <*ipv4-address*> variable displays debug information for a specific IPv4 host IP address.

The <*ipv6-address*> variable displays debug information for a specific IPv6 host IP address.

Use the **undebug** command to disable debugging.

Chapter

Access Control Lists 5

In this chapter

- In this chapter 59
- ACL entries and the Layer 4 CAM 63
- Configuring rule-based ACLs 65
- Modifying rule-based ACLs 73
- Displaying rule-based ACL entries 78
- ACL logging 82
- Dropping all fragments that exactly match a flow-based ACL 85
- Enabling ACL filtering of fragmented packets 86
- Enabling filtering for packets denied by flow-based ACLs 88
- Enabling strict TCP or UDP mode for flow-based ACLs 88
- ACLs and ICMP 91
- Using flow-based ACLs and NAT on the same interface 95
- Troubleshooting rule-based ACLs 96

How ServerIron ADX ADX processes ACLs

This chapter describes the access control list (ACL) feature. ACLs allow you to filter traffic based on the information in the IP packet header. Depending on the Foundry device, the device may also support Layer 2 ACLs, which filter traffic based on Layer 2 MAC header fields.

You can use IP ACLs to provide input to other features such as distribution lists and rate limiting. When you use an ACL this way, use permit statements in the ACL to specify the traffic that you want to send to the other feature. If you use deny statements, the traffic specified by the deny statements is not supplied to the other feature.

There are two ways that IPv4 ACLs are processed in Foundry devices: in software (flow-based ACLs) and in hardware (rule-based ACLs). This processing differs depending on the software release that you are running. These differences are described in the following sections.

Prior to release 12.3.01

Prior to release 12.3.01, IPv4 ACLs were processed as described in the following:

For deny actions:

All deny packets are dropped in hardware.

For permit actions:

For pass-through traffic, packets are processed in hardware.

For Layer 4 through Layer 7 traffic, packets are forwarded to the barrel processors (BPs) and the BPs perform the ACL processing.

Beginning with release 12.3.01 and later

Beginning with release 12.3.01, IPv4 ACLs are processed as described in the following:

For deny actions:

All deny packets are dropped in hardware.

For permit actions:

For pass-through traffic, packets are processed in hardware.

For Layer 4 through 7 traffic, packets are processed in hardware and then forwarded to the barrel processors (BPs). The BPs do not take any action on the ACLs.

Backwards compatibility option:

You can use the **ip flow-based-acl-enable** command to provide backwards compatibility for IPv4 ACL processing. If this command is configured, Layer 4 through 7 traffic, packets are processed in hardware and then forwarded to the barrel processors (BPs) where the BPs also process the ACLs. This command is configured as shown in the following.

```
ServerIronADX(config)# ip flow-based-acl-enable
```

Syntax: ip flow-based-acl-enable

Rule-based ACLs

Some Foundry devices process the traffic that ACLs filter in hardware. This document refers to this type of ACLs as *rule-based ACLs*. These ACLs are programmed into hardware at startup or as a new ACL is entered.

Rule-based ACLs program the ACL entries you assign to an interface into Content Addressable Memory (CAM) space allocated for the port(s). Devices that use rule-based ACLs program the ACLs into the CAM entries and use these entries to permit or deny packets in the hardware, without sending the packets to the CPU for processing.

Rule-based ACLs are supported on physical interfaces VE interfaces and trunk groups.

Types of rule-based ACLs

Rule-based ACLs can be configured as *standard* or *extended* ACLs.

- A standard ACL permits or denies packets based on source IP address.
- An extended ACL permits or denies packets based on source and destination IP address and also based on IP protocol information.

Both standard or extended ACLs can be *numbered* or *named*.

- Standard numbered ACLs have an ID of 1 through 99. Extended numbered ACLs are numbered 100 through 199.
- In this document, ACLs with a character string ID are called *named ACLs*. The IDs for both standard and extended ACLs can be character strings.

Configuration guidelines for rule-based ACLs

Consider the following guidelines:

- Rule-based ACLs support only one ACL per port. The ACL can contain multiple entries (rules). For example, rule-based ACLs do not support ACLs 101 and 102 on port 1, but rule-based ACLs do support ACL 101 containing multiple entries.
- If you change the content of an ACL (add, change, or delete entries), you must remove and then reapply the ACL to all the ports that use it. Otherwise, the older version of the ACL remains in the CAM and continues to be used. You can easily re-apply ACLs using the **ip rebind-acl** <*num*> | <*name*> | **all** command. Refer to "Applying ACLs to interfaces" on page 75.

NOTE
Foundry recommends that you also remove and reapply a changed ACL.

- ACL statistics are not supported with rule-based rate limiting.
- If you use the <*icmp-type*> parameter with an extended ACL, the device uses the CPU to filter packets using the ACL. The CPU is required to filter the ICMP message type.
- You can use policy-based routing (PBR) and rule-based ACLs on the same port. However, Foundry recommends that you use exactly the same ACL for each feature. Otherwise, it is possible for the ACL's Layer 4 CAM entry to be programmed incorrectly and give unexpected results.

Processing of fragmented packets

The descriptions for rule-based ACLs above apply to non-fragmented packets. The default processing of fragments by rule-based ACLs is as follows:

- The first fragment of a packet is permitted or denied using the ACLs. The first fragment is handled the same way as non-fragmented packets, since the first fragment contains the Layer 4 source and destination application port numbers. The device uses the Layer 4 CAM entry if one is programmed, or applies the interface's ACL entries to the packet and permits or denies the packet according to the first matching ACL.
- For other fragments of the same packet, one of the following occurs:
  - If the device has a CAM entry for the packet and has not been configured to send the fragments to the CPU, the device uses the CAM entry to forward the fragments in hardware.

    The fragments are forwarded even if the first fragment, which contains the Layer 4 information, was denied. Generally, denying the first fragment of a packet is sufficient, since a transaction cannot be completed without the entire packet. However, for stricter fragment control, you can send fragments to the CPU for filtering.
  - If the device is configured to send fragments to the CPU for filtering, the device compares the source and destination IP addresses to the ACL entries that contain Layer 4 information.
    - If the fragment's source and destination addresses exactly match an ACL entry that has Layer 4 information, the device assumes that the ACL entry is applicable to the fragment and permits or denies the fragment according to the ACL entry. The device does not compare the fragment to ACL entries that do not contain Layer 4 information.

- If both the fragment's source and destination addresses do not exactly match an ACL entry, the device skips the ACL entry and compares the packet to the next ACL entry. This is true even if either the source or destination address (but not both) does exactly match an ACL entry.
- If the source and destination addresses do not exactly match any ACL entry on the applicable interface, the device drops the fragment.

NOTE
By default, 10 Gigabit Ethernet modules also forward the first fragment instead of using the ACLs to permit or deny the fragment.

You can modify the handling of denied fragments. In addition, you can throttle the fragment rate on an interface that used rule-based ACLs. Refer to "Dropping all fragments that exactly match a flow-based ACL" on page 85 and "Enabling ACL filtering of fragmented packets" on page 86.

Default ACL action

The default action when no ACLs are configured on a device is to permit all traffic. However, once you configure an ACL and apply it to a port, the default action for that port is to deny all traffic that is not explicitly permitted on the port:

- If you want to tightly control access, configure ACLs consisting of permit entries for the access you want to permit. The ACLs implicitly deny all other access.
- If you want to secure access in environments with many users, you might want to configure ACLs that consist of explicit deny entries, then add an entry to permit all access to the end of each ACL. The software permits packets that are not denied by the deny entries.

ACL IDs and entries

ACLs consist of ACL IDs and ACL entries:

- ACL ID: An ACL ID is a number from 1 through 99 (for a standard ACL) or 100 through 199 (for an extended ACL) or a character string. The ACL ID identifies a collection of individual ACL entries. When you apply ACL entries to an interface, you do so by applying the ACL ID that contains the ACL entries to the interface, instead of applying the individual entries to the interface. This makes applying large groups of access filters (ACL entries) to interfaces simple.

NOTE
This is different from IP access policies. If you use IP access policies, you apply the individual policies to interfaces.

- ACL entry: An ACL entry are the filter commands associated with an ACL ID. These are also called *statements*. The maximum number of ACL entries you can configure is a system-wide parameter and depends on the device you are configuring. You can configure up to the maximum number of entries in any combination in different ACLs. The total number of entries in all ACLs cannot exceed the system maximum.

Layer 3 switch code on devices can support up to 8192 ACL entries.

You configure ACLs on a global basis, then apply them to the incoming or outgoing traffic on specific ports. You can apply only one ACL to a port's inbound traffic and only one ACL to a port's outbound traffic. The software applies the entries within an ACL in the order they appear in the ACL's configuration. As soon as a match is found, the software takes the action specified in the ACL entry (permit or deny the packet) and stops further comparison for that packet.

Support for up to 4096 ACL entries

You can configure up to 4096 ACL entries on devices that have enough space to hold a startup-config file that contains the ACLs.

For system-max configuration of 4096 ACL rules, the **ip access-group max-l4-cam** parameter must be set to 4096. To configure the maximum ACL rule limit of 4096 ACL rules, the following must be set:

1. The **system-max** for **ip-filter-sys** value must be set to 4096.

   ```
   ServerIronADX(config)# system-max ip-filter-sys 4096
   ```

2. The **ip access-group max-l4-cam** parameter must be set to 4096 on the interface that the ACL will be applied.

   ```
   ServerIronADX(config)# interface ethernet 1
   ServerIronADX(config-if-e1000-1)# ip access-group max-l4-cam 4096
   ```

3. Execute the **write memory** command to save the running configuration to the startup-config reload the ServerIron ADX ADX.

The actual number of ACLs you can configure and store in the startup-config file depends on the amount of memory available on the device for storing the startup-config. To store 4096 ACLs in the startup-config file requires at least 250K bytes, which is larger than the space available on a device's flash memory module.

You can load ACLs dynamically by saving them in an external configuration file on flash card or TFTP server, then loading them using one of the following commands.

```
copy tftp running-config <ip-addr> <filename>
ncopy tftp <ip-addr> <from-name> running-config
```

In this case, the ACLs are added to the existing configuration.

ACL entries and the Layer 4 CAM

Both standard and extended rule-based ACLs use Layer 4 CAM entries.

This section includes the following topics:

- "Aging out of entries in the Layer 4 CAM" on page 63
- "Displaying the number of Layer 4 CAM entries" on page 64
- "Specifying the maximum number of CAM entries" on page 64

Aging out of entries in the Layer 4 CAM

On a ServerIron ADX ADX device, the device permanently programs rule-based ACLs into the CAM. The entries never age out.

Displaying the number of Layer 4 CAM entries

To display the number of Layer 4 CAM entries used by each ACL, enter the **show access-list all** command.

```
ServerIronADX(config)# show access-list all

Extended IP access list 100 (Total flows: N/A, Total packets: N/A, Total rule cam
use: 3)
permit udp host 192.168.2.169 any (Flows: N/A, Packets: N/A, Rule cam use: 1)
permit icmp any any (Flows: N/A, Packets: N/A, Rule cam use: 1)
deny ip any any (Flows: N/A, Packets: N/A, Rule cam use: 1)
```

Syntax: show access-list <*acl-num*> | <*acl-name*> | **all**

The Rule cam use field lists the number of CAM entries used by the ACL or entry. The number of CAM entries listed for the ACL itself is equal to the total number of the CAM entries used by the ACL's entries.

Specifying the maximum number of CAM entries

For rule-based ACLs, you can adjust the allocation of Layer 4 CAM space for use by ACLs, on an IPC or IGC basis and on 10 Gigabit Ethernet modules. The new allocation applies to all the ports managed by the IPC or IGC or 10 Gigabit Ethernet module.

Most ACLs require one CAM entry for each ACL entry (rule). The exception is an ACL entry that matches on more than one TCP or UDP application port. In this case, the ACL entry requires a separate Layer 4 CAM entry for each application port on which the ACL entry matches.

Make sure you specify a maximum that is equal to or greater than the largest number of entries required by an ACL applied to any of the ports managed by the same IPC or IGC. For example, if port 1 will have an ACL that requires 250 entries, make sure 250 is the lowest number of entries you specify for any port on IPC 1 (the IPC that manages ports 1 through 24).

To specify the maximum number of CAM entries the device can allocate for rule-based ACLs, enter commands such as the following.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group max-l4-cam 50
```

This command allows up to 50 ACL entries on each port managed by the IPC or IGC that manages port 1/1.

Syntax: [**no**] **ip access-group max-l4-cam** <*num*>

The <*num*> parameter specifies the number of CAM entries and can be from 10 through 2048. The default value depends on the device.

The command is valid at the interface configuration level. However, the device applies the change to all ports managed by the same IPC or IGC. Regardless of the port number, when you save the change to the startup-config file, the CLI applies the command to the first port managed by the IPC or IGC. For example, if you enter the command on port 3, when you save the configuration change, the CLI enters the **ip access-group max-l4-cam** command under port 1 in the startup-config file.

NOTE
If you enter the **ip access-group max-l4-cam** command on more than one port managed by the same IPC or IGC, the CLI uses the value entered with the most-recent command for all the ports on the ICP or IGC.

Configuring rule-based ACLs

When you configure rule-based ACLs, you can refer to the ACL by a numeric ID or by an alphanumeric name. The commands to configure numbered ACLs are different from the commands for named ACLs:

- Numbered ACLs: If you refer to the ACL by a numeric ID, you can use the numbers 1 through 99 for a standard ACL or 100 through 199 for an extended ACL. In this document, this type of ACL is referred to as a *numbered ACL*. You can configure up to 100 standard numbered ACLs and 100 extended numbered ACLs.
- Named ACLs: If you refer to the ACL by a name, you must define the ACL as either a standard ACL or an extended ACL, and then specify the name of that ACL. In this document, this type of ACL is referred to as a *named ACL*. You can configure up to 100 standard named ACLs and 100 extended named ACLs by number.

No matter the total number of ACLs, the device supports a maximum of 1024 ACL entries, associated with the ACLs in any combination. (On ServerIron Chassis devices with Management 2 or Management 3 modules, the maximum is 2048.)

Configuring standard numbered ACLs

This section describes how to configure standard numbered ACLs with numeric IDs.

Standard ACLs permit or deny packets based on source IP address. You can configure up to 99 standard ACLs. There is no limit to the number of ACL entries an ACL can contain except for the system-wide limitation. For the number of ACL entries supported on a device, refer to "ACL IDs and entries" on page 62.

To configure a standard ACL and apply it to outgoing traffic on port 1/1, enter the following commands.

```
ServerIronADX(config)# access-list 1 deny host 209.157.22.26
ServerIronADX(config)# access-list 1 deny 209.157.29.12
ServerIronADX(config)# access-list 1 deny host IPHost1
ServerIronADX(config)# access-list 1 permit any
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group 1 in
ServerIronADX(config)# write memory
```

The commands in this example configure an ACL to deny packets from three source IP addresses from being forwarded on port 1/1. The last ACL entry in this ACL permits all packets that are not explicitly denied by the first three ACL entries.

Syntax: [**no**] **access-list** <*num*> **deny** | **permit** <*source-ip*> | <*hostname*> <*wildcard*>

or

Syntax: [**no**] **access-list** <*num*> **deny** | **permit** <*source-ip*>/<*mask-bits*> | <*hostname*>

Syntax: [**no**] **access-list** <*num*> **deny** | **permit host** <*source-ip*> | <*hostname*>

Syntax: [**no**] **access-list** <*num*> **deny** | **permit any**

Syntax: [**no**] **ip access-group** <*num*> **in** | **out**

The <*num*> parameter is the access list number and can be from 1 through 99.

The **deny** | **permit** parameter indicates whether packets that match a policy in the access list are denied (dropped) or permitted (forwarded).

The <*source-ip*> parameter specifies the source IP address. Alternatively, you can specify the host name.

NOTE
To specify the host name instead of the IP address, the host name must be configured using the Foundry device's DNS resolver. To configure the DNS resolver name, use the **ip dns server-address** command at the global CONFIG level of the CLI.

The <*wildcard*> parameter specifies the mask value to compare against the host address specified by the <*source-ip*> parameter. The <*wildcard*> is a four-part value in dotted-decimal notation (IP address format) consisting of ones and zeros. Zeros in the mask mean the packet's source address must match the <*source-ip*>. Ones mean any value matches. For example, the <*source-ip*> and <*wildcard*> values 209.157.22.26 0.0.0.255 mean that all hosts in the Class C subnet 209.157.22.x match the policy.

If you prefer to specify the wildcard (mask value) in CIDR format, you can enter a forward slash after the IP address, then enter the number of significant bits in the mask. For example, you can enter the CIDR equivalent of "209.157.22.26 0.0.0.255" as "209.157.22.26/24". The CLI automatically converts the CIDR number into the appropriate ACL mask (where zeros instead of ones are the significant bits) and changes the non-significant portion of the IP address into ones. For example, if you specify 209.157.22.26/24 or 209.157.22.26 0.0.0.255, then save the changes to the startup-config file, the value appears as 209.157.22.0/24 (if you have enabled display of subnet lengths) or 209.157.22.0 0.0.0.255 in the startup-config file.

If you enable the software to display IP subnet masks in CIDR format, the mask is saved in the file in "/<*mask-bits*>" format. To enable the software to display the CIDR masks, enter the **ip show-subnet-length** command at the global CONFIG level of the CLI. You can use the CIDR format to configure the ACL entry regardless of whether the software is configured to display the masks in CIDR format.

NOTE
If you use the CIDR format, the ACL entries appear in this format in the running-config and startup-config files, but are shown with subnet mask in the display produced by the **show ip access-lists** command.

The **host** <*source-ip*> | <*hostname*> parameter lets you specify a host IP address or name. When you use this parameter, you do not need to specify the mask. A mask of all zeros (0.0.0.0) is implied.

The **any** parameter configures the policy to match on all host addresses.

The **in** | **out** parameter specifies whether the ACL applies to incoming traffic or outgoing traffic on the interface to which you apply the ACL. You can apply the ACL to an Ethernet port. Note that the out option is not supported in the rule-based ACL mode.

Configuring extended numbered ACLs

This section describes how to configure extended numbered ACLs.

Extended ACLs let you permit or deny packets based on the following information:

- IP protocol
- Source IP address or host name
- Destination IP address or host name
- Source TCP or UDP port (if the IP protocol is TCP or UDP)
- Destination TCP or UDP port (if the IP protocol is TCP or UDP)

The IP protocol can be one of the following well-known names or any IP protocol number from 0 through 255:

- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- Internet Gateway Routing Protocol (IGRP)
- Internet Protocol (IP)
- Open Shortest Path First (OSPF)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

For TCP and UDP, you also can specify a comparison operator and port name or number. For example, you can configure a policy to block web access to a specific website by denying all TCP port 80 (HTTP) packets from a specified source IP address to the website's IP address.

To configure an extended access list that blocks all Telnet traffic received on port 1/1 from IP host 209.157.22.26, enter the following commands.

```
ServerIronADX(config)# access-list 101 deny tcp host 209.157.22.26 any eq telnet
ServerIronADX(config)# access-list 101 permit ip any any
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group 101 in
ServerIronADX(config)# write memory
```

Here is another example of commands for configuring an extended ACL and applying it to an interface. These examples show many of the syntax choices.

```
ServerIronADX(config)# access-list 102 permit icmp 209.157.22.0/24
209.157.21.0/24
ServerIronADX(config)# access-list 102 deny igmp host rkwong 209.157.21.0/24
ServerIronADX(config)# access-list 102 deny igrp 209.157.21.0/24 host rkwong
ServerIronADX(config)# access-list 102 deny ip host 209.157.21.100 host
209.157.22.1
ServerIronADX(config)# access-list 102 deny ospf any any
ServerIronADX(config)# access-list 102 permit ip any any
```

The first entry permits ICMP traffic from hosts in the 209.157.22.x network to hosts in the 209.157.21.x network.

The second entry denies IGMP traffic from the host device named "rkwong" to the 209.157.21.x network.

The third entry denies IGRP traffic from the 209.157.21.x network to the host device named “rkwong”.

The fourth entry denies all IP traffic from host 209.157.21.100 to host 209.157.22.1.

The fifth entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming or outgoing IP traffic on the ports to which you assign the ACL.

The following commands apply ACL 102 to the incoming traffic on port 1/2 and to the incoming traffic on port 4/3.

```
ServerIronADX(config)# int eth 1/2
ServerIronADX(config-if-1/2)# ip access-group 102 in
ServerIronADX(config-if-1/2)# exit
ServerIronADX(config)# int eth 4/3
ServerIronADX(config-if-4/3)# ip access-group 102 in
ServerIronADX(config)# write memory
```

Here is another example of an extended ACL.

```
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 209.157.22.0/24
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 eq ftp
209.157.22.0/24
ServerIronADX(config)# access-list 103 deny tcp 209.157.21.0/24 209.157.22.0/24 lt
telnet neq 5
ServerIronADX(config)# access-list 103 deny udp any range 5 6 209.157.22.0/24
range 7 8
ServerIronADX(config)# access-list 103 permit ip any any
```

The first entry in this ACL denies TCP traffic from the 209.157.21.x network to the 209.157.22.x network.

The second entry denies all FTP traffic from the 209.157.21.x network to the 209.157.22.x network.

The third entry denies TCP traffic from the 209.157.21.x network to the 209.157.22.x network, if the TCP port number of the traffic is less than the well-known TCP port number for Telnet (23), and if the TCP port is not equal to 5. Thus, TCP packets whose TCP port numbers are 5 or are greater than 23 are allowed.

The fourth entry denies UDP packets from any source to the 209.157.22.x network, if the UDP port number from the source network is 5 or 6 and the destination UDP port is 7 or 8.

The fifth entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming or outgoing IP traffic on the ports to which you assign the ACL.

The following commands apply ACL 103 to the incoming traffic on ports 2/1 and 2/2.

```
ServerIronADX(config)# int eth 2/1
ServerIronADX(config-if-2/1)# ip access-group 103 in
ServerIronADX(config-if-2/1)# exit
ServerIronADX(config)# int eth 2/2
ServerIronADX(config-if-2/2)# ip access-group 103 in
ServerIronADX(config)# write memory
```

Use the following syntax for configuring extended numbered ACLs.

Syntax: [**no**] **access-list** <*num*> **deny** | **permit** <*ip-protocol*> <*source-ip*> | <*hostname*> <*wildcard*> [<*operator*> <*source-tcp/udp-port*>] <*destination-ip*> | <*hostname*> [<*icmp-type*> | <*icmp-num*> | <*icmp-type-number*> <*icmp-code-number*>] <*wildcard*> [<*operator*> <*destination-tcp/udp-port*>] [**established**] [**precedence** <*name*> | <*num*>] [**tos** <*name*> | <*num*>] [**ip-pkt-len** <*value*>]

Syntax: [**no**] **access-list** <*num*> **deny** | **permit host** <*ip-protocol*> **any any**

Syntax: [**no**] **ip access-group** <*num*> **in** | **out**

The <*num*> parameter indicates the ACL number and be from 100 through 199 for an extended ACL.

The **deny** | **permit** parameter indicates whether packets that match the policy are dropped or forwarded.

The <*ip-protocol*> parameter indicates the type of IP packet you are filtering. You can specify a well-known name for any protocol whose number is less than 255. For other protocols, you must enter the number. Enter "?" instead of a protocol to list the well-known names recognized by the CLI.

The <*source-ip*> | <*hostname*> parameter specifies the source IP host for the policy. If you want the policy to match on all source addresses, enter **any**.

The <*wildcard*> parameter specifies the portion of the source IP host address to match against. The <*wildcard*> is a four-part value in dotted-decimal notation (IP address format) consisting of ones and zeros. Zeros in the mask mean the packet's source address must match the <*source-ip*>. Ones mean any value matches. For example, the <*source-ip*> and <*wildcard*> values 209.157.22.26 0.0.0.255 mean that all hosts in the Class C subnet 209.157.22.x match the policy.

If you prefer to specify the wildcard (mask value) in Classless Interdomain Routing (CIDR) format, you can enter a forward slash after the IP address, then enter the number of significant bits in the mask. For example, you can enter the CIDR equivalent of "209.157.22.26 0.0.0.255" as "209.157.22.26/24". The CLI automatically converts the CIDR number into the appropriate ACL mask (where zeros instead of ones are the significant bits) and changes the non-significant portion of the IP address into zeros. For example, if you specify 209.157.22.26/24 or 209.157.22.26 0.0.0.255, then save the changes to the startup-config file, the value appears as 209.157.22.0/24 (if you have enabled display of subnet lengths) or 209.157.22.0 0.0.0.255 in the startup-config file.

If you enable the software to display IP subnet masks in CIDR format, the mask is saved in the file in "/<*mask-bits*>" format. To enable the software to display the CIDR masks, enter the **ip show-subnet-length** command at the global CONFIG level of the CLI. You can use the CIDR format to configure the ACL entry regardless of whether the software is configured to display the masks in CIDR format.

NOTE
If you use the CIDR format, the ACL entries appear in this format in the running-config and startup-config files, but are shown with subnet mask in the display produced by the **show ip access-lists** command.

The <*destination-ip*> | <*hostname*> parameter specifies the destination IP host for the policy. If you want the policy to match on all destination addresses, enter **any**.

The <*icmp-type*> | <*icmp-num*> parameter specifies the ICMP protocol type.

- This parameter applies only if you specified **icmp** as the <*ip-protocol*> value.
- If you use this parameter, the ACL entry is sent to the CPU for processing.

- If you do not specify a message type, the ACL applies to all types of ICMP messages.

The <*icmp-num*> parameter can be a value from 0 through 255.

The <*icmp-type*> parameter can have one of the following values, depending on the software version the device is running:

- any-icmp-type
- echo
- echo-reply
- information-request
- log
- mask-reply
- mask-request
- parameter-problem
- redirect
- source-quench
- time-exceeded
- timestamp-reply
- timestamp-request
- unreachable
- <*num*>

The <*operator*> parameter specifies a comparison operator for the TCP or UDP port number. This parameter applies only when you specify **tcp** or **udp** as the IP protocol. For example, if you are configuring an entry for HTTP, specify **tcp eq http**. You can enter one of the following operators:

- **eq** : The policy applies to the TCP or UDP port name or number you enter after the the **eq** operand.
- **gt**: The policy applies to TCP or UDP port numbers greater than the port number or the numeric equivalent of the port name you enter after the **gt** operand.
- **lt**: The policy applies to TCP or UDP port numbers that are less than the port number or the numeric equivalent of the port name you enter after the **lt** operand.
- **neq**: The policy applies to all TCP or UDP port numbers except the port number or port name you enter after the **neq** operand.
- **range**: The policy applies to all TCP or UDP port numbers that are between the first TCP or UDP port name or number and the second one you enter following the range parameter. The range includes the port names or numbers you enter. For example, to apply the policy to all ports between and including 23 (Telnet) and 53 (DNS), enter the following: **range 23 53**. The first port number in the range must be lower than the last number in the range.
- **established**: This operator applies only to TCP packets. If you use this operator, the policy applies to TCP packets that have the ACK (Acknowledgment) or RST (Reset) bits set on (set to "1") in the Control Bits field of the TCP packet header. Thus, the policy applies only to established TCP sessions, not to new sessions. Refer to Section 3.1, "Header Format", in RFC 793 for information about this field.

NOTE
This operator applies only to destination TCP ports, not source TCP ports.

The <*tcp/udp-port*> parameter specifies the TCP or UDP port number or well-known name. You can specify a well-known name for any application port whose number is less than 1024. For other application ports, you must enter the number. Enter "?" instead of a port to list the well-known names recognized by the CLI.

The **in** | **out** parameter specifies whether the ACL applies to incoming traffic or outgoing traffic on the interface to which you apply the ACL. You can apply the ACL to an Ethernet port.

NOTE
The **out** option is not supported in the rule-based ACL mode.

The **precedence** <*name*> | <*num*> parameter of the **ip access-list** command specifies the IP precedence. The precedence option for of an IP packet is set in a three-bit field following the four-bit header-length field of the packet's header. You can specify one of the following:

- **critical** or **5**: The ACL matches packets that have the critical precedence. If you specify the option number instead of the name, specify number 5.
- **flash** or **3**: The ACL matches packets that have the flash precedence. If you specify the option number instead of the name, specify number 3.
- **flash-override** or **4**: The ACL matches packets that have the flash override precedence. If you specify the option number instead of the name, specify number 4.
- **immediate** or **2**: The ACL matches packets that have the immediate precedence. If you specify the option number instead of the name, specify number 2.
- **internet** or **6**: The ACL matches packets that have the internetwork control precedence. If you specify the option number instead of the name, specify number 6.
- **network** or **7**: The ACL matches packets that have the network control precedence. If you specify the option number instead of the name, specify number 7.
- **priority** or **1**: The ACL matches packets that have the priority precedence. If you specify the option number instead of the name, specify number 1.
- **routine** or **0**: The ACL matches packets that have the routine precedence. If you specify the option number instead of the name, specify number 0.

The **tos** <*name*> | <*num*> parameter of the **ip access-list** command specifies the IP ToS. You can specify one of the following:

- **max-reliability** or **2**: The ACL matches packets that have the maximum reliability ToS. The decimal value for this option is 2.
- **max-throughput** or **4**: The ACL matches packets that have the maximum throughput ToS. The decimal value for this option is 4.
- **min-delay** or **8**: The ACL matches packets that have the minimum delay ToS. The decimal value for this option is 8.
- **min-monetary-cost** or **1**: The ACL matches packets that have the minimum monetary cost ToS. The decimal value for this option is 1.

  NOTE
  This value is not supported on 10 Gigabit Ethernet modules.

- **normal** or **0**: The ACL matches packets that have the normal ToS. The decimal value for this option is 0.

- <*num*>: A number from 0 through 15 that is the sum of the numeric values of the options you want. The ToS field is a four-bit field following the Precedence field in the IP header. You can specify one or more of the following. To select more than one option, enter the decimal value that is equivalent to the sum of the numeric values of all the ToS options you want to select. For example, to select the **max-reliability** and **min-delay** options, enter number 10. To select all options, select 15.

The **ip-pkt-len** <*value*> parameter filters ICMP packets based on the IP packet length. The device uses the <*value*> to match the total length field in the IP header of ICMP packets. You can specify a value from 1 through 65535.

NOTE
This parameter applies only if you specified **icmp** as the <*ip-protocol*> value.

The **log** parameter enables SNMP traps and syslog messages for packets denied by the ACL.

You can enable logging on ACLs and filters that support logging even when the ACLs and filters are already in use. To do so, re-enter the **ACL** or **filter** command and add the **log** parameter to the end of the ACL or filter. The software replaces the **ACL** or **filter** command with the new one. The new ACL or filter, with logging enabled, takes effect immediately.

Configuring standard or extended named ACLs

The configuration of named ACLs differs from the configuration of numbered ACLs in two significant ways:

- Named ACL entries are configured using the **ip access-list** command. (Numbered ACL entries, on the other hand, are configured using the **access-list**. command.)
- When you configure a named ACL, you must first specify the ACL type (standard or extended) and the ACL name with one command, which places you in the configuration level for that ACL. (Numbered ACL entries, on the other hand, may be configured on a single line.) Otherwise, the command syntax for specifying named and numbered ACLs is the same.

The following configuration examples demonstrate the configuration of standard named ACLs and extended named ACLs.

Configuration example for standard named ACL

To configure a named standard ACL entry, enter commands such as the following.

```
ServerIronADX(config)# ip access-list standard Net1
ServerIronADX(config-std-nacl)# deny host 209.157.22.26 log
ServerIronADX(config-std-nacl)# deny 209.157.29.12 log
ServerIronADX(config-std-nacl)# deny host IPHost1 log
ServerIronADX(config-std-nacl)# permit any
ServerIronADX(config-std-nacl)# exit
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group Net1 out
```

The commands in this example configure a standard ACL named “Net1”. The entries in this ACL deny packets from three source IP addresses from being forwarded on port 1/1. Since the implicit action for an ACL is “deny”, the last ACL entry in this ACL permits all packets that are not explicitly denied by the first three ACL entries. For an example of how to configure the same entries in a numbered ACL, refer to “Configuring standard numbered ACLs” on page 65.

Notice that the command prompt changes after you enter the ACL type and name. The “std” in the command prompt indicates that you are configuring entries for a standard ACL. For an extended ACL, this part of the command prompt is “ext“. The “nacl” indicates that are configuring a named ACL.

Syntax: ip access-list extended | **standard** <*acl-name*> | <*acl-num*>

The **extended** | **standard** parameter indicates the ACL type.

The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”). The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, you can specify from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.

NOTE
For convenience, the software allows you to configure numbered ACLs using the syntax for named ACLs. The software also still supports the older syntax for numbered ACLs. Although the software allows both methods for configuring numbered ACLs, numbered ACLs are always formatted in the startup-config and running-config files in using the older syntax, as follows.

```
access-list 1 deny host 209.157.22.26
access-list 1 deny 209.157.22.0 0.0.0.255
access-list 1 permit any
access-list 101 deny tcp any any eq http
```

The options at the ACL configuration level and the syntax for the **ip access-group** command are the same for numbered and named ACLs and are described in “Configuring standard numbered ACLs” on page 65.

Configuration example for extended named ACL

To configure a named extended ACL entry, enter commands such as in the following example.

```
ServerIronADX(config)# ip access-list extended “block Telnet”
ServerIronADX(config-ext-nacl)# deny tcp host 209.157.22.26 any eq telnet
ServerIronADX(config-ext-nacl)# permit ip any any
ServerIronADX(config-ext-nacl)# exit
ServerIronADX(config)# int eth 1/1
ServerIronADX(config-if-1/1)# ip access-group “block Telnet” in
```

The options at the ACL configuration level and the syntax for the **ip access-group** command are the same for numbered and named ACLs and are described in “Configuring extended numbered ACLs” on page 67.

Modifying rule-based ACLs

This section includes the following topics:

- “Reordering ACLs” on page 74
- “Applying ACLs to interfaces” on page 75

Reordering ACLs

When you use the Foundry device's CLI to configure any ACL, the software places the ACL entries in the ACL in the order you enter them. For example, if you enter the following entries in the order shown below, the software always applies the entries to traffic in the same order.

```
ServerIronADX(config)# access-list 1 deny 209.157.22.0/24
ServerIronADX(config)# access-list 1 permit 209.157.22.26
```

If a packet matches the first ACL entry in this ACL and is therefore denied, the software does not compare the packet to the remaining ACL entries. In this example, packets from host 209.157.22.26 will always be dropped, even though packets from this host match the second entry.

You can use the CLI to reorder entries within an ACL by individually removing the ACL entries and then re-adding them. To use this method, enter the **no** operand followed by the command for an ACL entry, and repeat this for each ACL entry in the ACL you want to edit. After removing all the ACL entries from the ACL, re-add them.

This method works well for small ACLs such as the example above, but can be impractical for ACLs containing many entries. Therefore, Foundry devices provide an alternative method. The alternative method lets you upload an ACL list from a TFTP server and replace the ACLs in the device's running-config file with the uploaded list. Thus, to change an ACL, you can edit the ACL on the file server, then upload the edited ACL to the device. You then can save the changed ACL to the device's startup-config file.

ACL lists contain only the ACL entries themselves, not the assignments of ACLs to interfaces. You must assign the ACLs on the device itself.

NOTE
The only valid commands that are valid in the ACL list are the **access-list** and **end** commands. The Foundry device ignores other commands in the file.

Use the following procedure to modify an ACL by configuring an ACL list on a file server.

1. Use a text editor to create a new text file. When you name the file, use 8.3 format (up to eight characters in the name and up to three characters in the extension).

   NOTE
   Make sure the Foundry device has network access to the TFTP server.

2. Optionally, clear the ACL entries from the ACLs you are changing by placing commands such as the following at the top of the file.

   ```
   no access-list 1
   no access-list 101
   ```

   When you load the ACL list into the device, the software adds the ACL entries in the file after any entries that already exist in the same ACLs. Thus, if you intend to entirely replace an ACL, you must use the **no access-list** <*num*> command to clear the entries from the ACL before the new ones are added.

3. Place the commands to create the ACL entries into the file. The order of the separate ACLs does not matter, but the order of the entries within each ACL is important. The software applies the entries in an ACL in the order they are listed within the ACL. Here is an example of some ACL entries.

```
access-list 1 deny host 209.157.22.26 log
access-list 1 deny 209.157.22.0 0.0.0.255 log
access-list 1 permit any
access-list 101 deny tcp any any eq http log
```

The software will apply the entries in ACL 1 in the order shown and stop at the first match. Thus, if a packet is denied by one of the first three entries, the packet will not be permitted by the fourth entry, even if the packet matches the comparison values in this entry.

4. Enter the command **end** on a separate line at the end of the file. This command indicates to the software that the entire ACL list has been read from the file.
5. Save the text file.
6. On the Foundry device, enter the following command at the Privileged EXEC level of the CLI.

```
copy tftp running-config <tftp-ip-addr> <filename>
```

NOTE
This command will be unsuccessful if you place any commands other than access-list and end (at the end only) in the file. These are the only commands that are valid in a file you load using the **copy tftp running-config** command.

7. To save the changes to the device's startup-config file, enter the following command at the Privileged EXEC level of the CLI.

```
write memory
```

Here is a complete example of an ACL configuration file.

```
no access-list 1
no access-list 101
access-list 1 deny host 209.157.22.26 log
access-list 1 deny 209.157.22.0 0.0.0.255 log
access-list 1 permit any
access-list 101 deny tcp any any eq http log
end
```

NOTE
Do not place other commands in the file. The Foundry device reads only the ACL information in the file and ignores other commands, including **ip access-group** commands. To assign ACLs to interfaces, use the CLI.

Applying ACLs to interfaces

Configuration examples in the section "Configuring rule-based ACLs" on page 65 show that you apply ACLs to interfaces using the **ip access-group** command.

If you make an ACL configuration change, you must reapply the ACLs to their interfaces to place the change into effect.

An ACL configuration change includes any of the following:

- Adding, changing, or removing an ACL or an entry in an ACL
- Changing a PBR policy

To reapply ACLs following an ACL configuration change, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip rebind-acl all
```

Syntax: [**no**] **ip rebind-acl** <*num*> | <*name*> | **all**

Adding, replacing, or deleting comments to rule-based ACLs

The **remark** subcommand enables you to include comments about entries in access control lists (ACLs). Comments make it easier for network administrators to scan and understand ACL entries.

You can add, replace, or delete comments applied to both numbered and named access control lists using the **remark** subcommand with either the **access-list** command (for numbered ACLs) or **ip access-list** command (for named ACLs).

In general, a comment should be entered immediately before the ACL entry it defines. The comment must be entered separately from the actual ACL entry; that is, you cannot enter the ACL entry and the ACL comment with the same command. If the comment is not entered immediately before the ACL entry it describes, the comment will not be displayed correctly in the output of the **show access-list** or **show ip access-lists** commands.

Adding comments to numbered ACLs

To add a comment to a numbered ACL, enter the **access-list** <*acl-num*> **remark** <*comment-text*> command immediately before you define an ACL entry such as in the following example.

```
ServerIronADX(config)# access-list 99 remark Permit all users
ServerIronADX(config)# access-list 99 permit all
```

The first line defines the comment. The second line defines an entry in the numbered ACL called *99*. For the comment to be displayed correctly in the output of **show** commands, the remark must be entered immediately before the ACL entry it describes.

Syntax: **access-list** <*acl-num*> **remark** <*comment-text*>

The **remark** <*comment-text*> subcommand adds a comment to the ACL entry. The remark be up to 128 characters in length. The comment must be entered separately from the actual ACL entry.

Replacing comments applied to numbered ACLs

To replace the last comment entered for a numbered ACL, enter the **access-list** <*acl-num*> **remark** <*comment-text*> command. The previous comment is overwritten.

For example, to replace an existing comment applied to a numbered ACL, enter a command such as the following.

```
ServerIronADX(config)# access-list 99 remark Permit marketing
```

The last comment entered for the ACL called *99* is overwritten with the comment “Permit marketing”.

NOTE
Only the last comment applied to the ACL can be replaced using this method. If multiple comments have been applied to the ACL, you must delete and reenter the ACL entry to update the comment applied to that ACL entry.

Deleting comments applied to numbered ACLs

To delete a comment from a numbered ACL, enter the **access-list** <*acl-num*> **remark** <*comment-text*> command using the **no** operand such as in the following example.

```
ServerIronADX(config)# no access-list 99 remark Permit all users
```

In the example, the command deletes the comment "Permit all users" from the ACL. The value entered for the <*comment-text*> parameter must exactly match that of the comment you want to delete.

Syntax: no access-list <*acl-num*> **remark** <*comment-text*>

The **no** operand indicates that the comment is to be deleted.

The <*acl-num*> parameter identifies the numbered ACL in which the comment appears.

Adding comments to named ACLs

To add a comment to a named ACL, enter the **remark** <*comment-text*> command immediately before you define an ACL entry such as in the following example.

```
ServerIronADX(config)# ip access-list standard melon
ServerIronADX(config-std-nacl)# remark Deny traffic from Marketing
ServerIronADX(config-std-nacl)# deny 5.6.7.8
```

In the example, the first command specifies a named standard ACL called *melon*, the second command defines the comment applied to the named ACL, and the third command defines an entry.

NOTE
The comment is entered immediately prior to the ACL entry.

Syntax: ip access-list standard | **extended** <*acl-name*> | <*acl-num*>

Syntax: remark <*comment-text*>

Syntax: deny <*options*> | **permit** <*options*>

The **standard** | **extended** parameter indicates the ACL type.

Named ACLs can be identified by either an <*acl-name*> or an <*acl-number*> value.

- The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, "ACL for Net1").
- The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.

The **remark** <*comment-text*> adds a comment to the ACL entry that you are about to create. The comment can have up to 128 characters in length. For the remark to be displayed correctly in the output of the **show access-list** and **show ip access-lists** commands, the comment must be entered immediately before the ACL entry it describes.

Enter the **deny** operand to deny the specified traffic or the **permit** operand to allow the specified traffic. Complete the configuration by specifying <*options*> for the standard or extended ACL entry.

Replacing comments applied to named ACLs

To replace the last comment entered for a named ACL, enter the **remark** <*comment-text*> command within the ACL configuration. The previous comment is overwritten.

In the following example, the last comment entered for the named ACL called *melon* is replaced with the comment “Permit sales”.

```
ServerIronADX(config)# ip access-lists standard melon
ServerIronADX(config-std-nacl)# remark Permit sales
```

NOTE
Only the last comment applied to the ACL can be replaced using this method. If multiple comments have been applied to the ACL, you must delete and reenter the ACL entry to update the comment applied to that ACL entry.

Deleting comments applied to named ACLs

To delete a comment from a named ACL, enter **remark** <*comment-text*> command using the **no** operand such as in the following example.

```
ServerIronADX(config)# ip access-list melon
ServerIronADX(config-std-nacl)# no remark All users allowed
```

Syntax: no remark <*comment-text*>

The **remark** <*comment-text*> identifies the comment to be deleted from the ACL entry. The <*comment-text*> must exactly match the existing value for the specified comment to be deleted.

Displaying rule-based ACL entries

The **show access-list** and **show ip access-lists** commands enable you to view the contents of named or numbered ACLs.

You can also use keywords with the **show access-list** command to filter the ACL entries displayed in command output to those that match a specified keyword. Keywords may include either numbers or characters.

This section includes the following topics:

- “Displaying ACLs using the show access-list command” on page 78
- “Displaying ACLs using the show ip access-lists command” on page 79
- “Displaying ACLs using keywords” on page 79
- “Displaying ACL bindings” on page 82

Displaying ACLs using the show access-list command

The **show access-list** command enables you to display the contents of numbered or named ACLs.

To display the contents of an ACL, enter a command such as the following.

```
ServerIronADX# show access-list 99
Standard IP access list 99
deny host 1.2.4.5
deny host 5.6.7.8
permit any
```

Syntax: show access-list <*acl-num*> | <*acl-name*> | **all** [**bindings**]

Access control lists can be identified by either an <*acl-name*> or an <*acl-number*> value. Numbered ACLs are always identified by a <*acl-num*> value. Named ACLs may be identified by either an <*acl-num*> or an <*acl-name*> value.

- The <*acl-num*> parameter allows you to specify an ACL number. If you specify a number, enter a number from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.
- The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”).

The **all** operand returns the contents of all numbered and named ACLs.

The **bindings** operand returns information about which ACLs are bound to which interfaces. For more information, refer to “Displaying ACL bindings” on page 82.

Displaying ACLs using the show ip access-lists command

The **show ip access lists** command enables you to display the contents of numbered or named ACLs.

To display the contents of an ACL, enter a command such as the following.

```
ServerIronADX# show ip access-lists melon
Standard IP access list melon
deny host 1.2.4.5
deny host 5.6.7.8
permit any
```

Syntax: show ip access-lists <*acl-num*> | <*acl-name*>

Access control lists can be identified by either an <*acl-name*> or an <*acl-number*> value. Numbered ACLs are always identified by a <acl-num> value. Named ACLs may be identified by either an <*acl-num*> or an <*acl-name*> value.

- The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.
- The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”).

Displaying ACLs using keywords

You can use keywords to limit the ACL entries returned for the **show access-list** command to those that match the keyword specified. Keywords may be used to filter both numbered and named ACLs.

A keyword may be either a numerical or text string.

Displaying ACLs using numerical keywords

Using numerical keywords you can choose to view only those ACL entries that match a specified numerical value, which can be useful for filtering ACL entries by the IP addresses they govern.

For example, consider an numbered ACL (*99*) that includes multiple entries. Entering the **show access-list** command will return all of the entries.

```
ServerIronADX(config)# show access-list 99
Standard IP access-list 99
deny host 1.2.3.4
deny host 2.3.4.5
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

If you want to display only those ACL entries that begin with a specified numerical keyword, enter a command using the **begin** <*keyword*> parameter such as the one in the following example.

```
ServerIronADX(config)# show access-list 99 | begin 5
Standard IP access-list 99
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

The command displays only those ACL entries that begin with the keyword “5”. ACL entries that do not begin with this keyword are not displayed.

If you want to display all of the ACL entries that contain a numerical keyword regardless of position, enter a command using the **include** <*keyword*> parameter such as the one in the following example.

```
ServerIronADX(config)#show access-list 99 | include 5
Standard IP access-list 99
deny host 2.3.4.5
permit host 5.6.7.8
permit host 5.10.11.12
```

All of the ACL entries in the ACL that contain the keyword “5” and are displayed.

If you want to exclude ACL entries that contain a specified keyword, enter a command using the **exclude** <*keyword*> parameter such as the one in the following example.

```
ServerIronADX(config)# show access-list 99 | exclude 5
Standard IP access-list 99
deny host 1.2.3.4
permit any
```

Because the second, third, and fourth ACL entries contain the keyword “5”, they are not displayed.

Syntax: show access-list <*acl-number*>| <*acl-name*>**| begin** | **exclude** | **include** <*keyword*>

The <*acl-num*> parameter allows you to specify an ACL number. If you specify a number, enter a number from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.

The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”).

Use the **|** operator to indicate a keyword.

Enter the **begin** <*keyword*> parameter to start the display beginning with the first line containing the text that matches the keyword. For example, if you enter “begin 5”, the displayed information begins with the line containing the number “5”.

Enter the **exclude** <*keyword*> parameter to exclude any lines containing text that match the keyword. For example, if you enter “exclude 5”, any line containing the number “5” is excluded from the display.

Enter the **include** <*keyword*> display only those lines containing text that match the keyword. For example, if you enter “include 5”, any line containing the number “5” is included in the display.

Displaying named ACLs using text string keywords

Using text string keywords you can choose to view only those ACL entries that match a specified text string, which can be useful if you want to view only those entries that permit or deny access.

Consider an access control list called *melon* that includes multiple entries. If you enter the **show access-list** command and specifiy the appropriate <*acl-name*> value, all of the ACL entries for the named ACL are returned.

```
ServerIronADX(config)# show access-list melon
Standard IP access-list melon
deny host 1.2.3.4
permit host 5.6.7.8
permit host 5.10.11.12
permit any
```

To display only those ACL entries that contain a specified text-string, enter a command such the one in the following example.

```
ServerIronADX(config)#show access-list 99 | include deny
Standard IP access-list 99
   ACL Remark: Deny Building A
deny host 1.2.3.4
deny host 5.10.11.12
```

In the example, only those ACL entries that contain the keyword “deny” are returned. Note that the **show access-list** command returns all ACL entries and comments that include the keyword “deny”.

Syntax: show ip access-lists <*acl-number*>| <*acl-name*>**| begin** | **exclude** | **include** <*keyword*>

The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 1 through 99 for standard ACLs or 100 through 199 for extended ACLs.

The <*acl-name*> parameter is the ACL name. You can specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”).

Use the **|** operator to indicate a keyword.

Enter the **begin** <*keyword*> parameter to start the display beginning with the first line containing text that matches the keyword. For example, if you enter “begin Permit”, the displayed information begins with the line containing the word “permit”.

Enter the **exclude** <*keyword*> parameter to exclude any lines containing text that match the keyword. For example, if you enter “exclude Permit”, any line containing the word “permit” is excluded from the display.

Enter the **include** <*keyword*> display only those lines containing text that match the keyword. For example, if you enter “include Permit”, any line containing the word “permit” is included in the display.

Displaying ACL bindings

To view which ACLs (IPv4 and IPv6) are bound to which interfaces, enter the **show access-list** command using the **bindings** keyword such as that shown in the following example.

```
ServerIronADX# show access-list bindings
Access-list binding configuration:
!
interface ethernet 2
ip access-group 2 in
ipv6 traffic-filter acl1 in
!
interface ve 2
ip access-group 111 in
ipv6 traffic-filter acl2 out
```

Syntax: show access-list bindings

ACL logging

You may want the software to log entries for ACLs in the syslog. This section present the how logging is processed by rule-based ACLs.

Rule-based ACLs do not support the **log** option. Even when rule-based ACLs are enabled, if an ACL entry has the **log** option, traffic that matches that ACL is sent to the CPU for processing. Depending on how many entries have the log option and how often packets match those entries, ACL performance can be affected.

If your configuration already contains ACLs that you want to use with rule-based ACLs, but some of the ACLs contain the **log** option, the software changes the ACL mode to flow-based for the traffic flows that match the ACL. Changing the mode to flow-based enables the device to send the matching flows to the CPU for processing. This is required because the CPU is needed to generate the syslog message.

You can globally disable ACL logging without the need to remove the **log** option from each ACL entry. When you globally disable ACL logging, the ACL entries remain unchanged but the **log** option is ignored and the ACL can use the rule-based ACL mode. This enables you to use the ACLs in the rule-based ACL mode. You also can configure the device to copy traffic that is denied by a rule-based ACL to an interface. This option allows you to monitor the denied traffic without sending the traffic to the CPU.

To globally disable ACL logging, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip access-list disable-log-to-cpu
```

Syntax: [**no**] **ip access-list disable-log-to-cpu**

To re-enable ACL logging, enter the following command.

```
ServerIronADX(config)# no ip access-list disable-log-to-cpu
```

Syslog message for changed ACL mode

If the device changes the ACL mode from rule-based to flow-based, the device generates one of the following syslog notification messages:

- ACL insufficient L4 session resource, using flow-based ACL instead.
- ACL exceed max DMA L4 CAM resource, using flow-based ACL instead. Refer to "Specifying the maximum number of CAM entries" on page 64.
- ACL insufficient L4 CAM resource, using flow-based ACL instead.

Copying denied traffic to a mirror port for monitoring

Although rule-based ACLs do not support ACL logging, you nonetheless can monitor the traffic denied by rule-based ACLs. To do so, attach a protocol analyzer to a port and enable the device to redirect traffic denied by ACLs to that port.

To redirect traffic denied by ACLs, enter the following command at the interface configuration level.

```
ServerIronADX(config-if-1/1)# ip access-group redirect-deny-to-interf
```

Syntax: [no] ip access-group redirect-deny-to-interf

Enter the command on the port to which you want the denied traffic to be copied.

NOTE
The software requires that an ACL has already been applied to the interface.

When you enable redirection, the deny action of the ACL entry is still honored. Traffic that matches the ACL is not forwarded.

Displaying ACL log entries

The first time an entry in an ACL permits or denies a packet and logging is enabled for that entry, the software generates a syslog message and an SNMP trap. Messages for packets permitted or denied by ACLs are at the warning level of the syslog.

When the first syslog entry for a packet permitted or denied by an ACL is generated, the software starts an ACL timer. After this, the software sends syslog messages every one to ten minutes, depending on the value of the timer interval. If an ACL entry does not permit or deny any packets during the timer interval, the software does not generate a syslog entry for that ACL entry.

NOTE
For an ACL entry to be eligible to generate a syslog entry for permitted or denied packets, logging must be enabled for the entry. The syslog contains entries only for the ACL entries that deny packets and have logging enabled.

To display syslog entries, enter the following command from any CLI prompt.

```
ServerIronADX(config)# show log
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Buffer logging: level ACDMEINW, 38 messages logged
    level code: A=alert C=critical D=debugging M=emergency E=error
                I=informational N=notification W=warning

Log Buffer (50 entries):

21d07h02m40s:warning:list 101 denied tcp 209.157.22.191(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
00d07h03m30s:warning:list 101 denied tcp 209.157.22.26(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
00d06h58m30s:warning:list 101 denied tcp 209.157.22.198(0)(Ethernet 4/18
0010.5a1f.77ed) -> 198.99.4.69(http), 1 event(s)
```

In this example, the two-line message at the bottom is the first entry, which the software immediately generates the first time an ACL entry permits or denies a packet. In this case, an entry in ACL 101 denied a packet. The packet was a TCP packet from host 209.157.22.198 and was destined for TCP port 80 (HTTP) on host 198.99.4.69.

When the software places the first entry in the log, the software also starts the five-minute timer for subsequent log entries. Thus, five minutes after the first log entry, the software generates another log entry and SNMP trap for denied packets.

In this example, the software generates the second log entry five minutes later.

The time stamp for the third entry is much later than the time stamps for the first two entries. In this case, no ACLs denied packets for a very long time. In fact, since no ACLs denied packets during the five-minute interval following the second entry, the software stopped the ACL log timer. The software generated the third entry as soon as the ACL denied a packet. The software restarted the five-minute ACL log timer at the same time. As long as at least one ACL entry permits or denies a packet, the timer continues to generate new log entries and SNMP traps every five minutes.

You can also configure the maximum number of ACL-related log entries that can be added to the system log over a one-minute period. For example, to limit the device to 100 ACL-related syslog entries per minute.

```
ServerIronADX(config)# max-acl-log-num 100
```

Syntax: [**no**] **max-acl-log-num** <*num*>

You can specify a number between 0 through 4096. The default is 256. Specifying 0 disables all ACL logging.

Displaying ACL statistics for flow-based ACLs

To display ACL statistics for flow-based ACLs, enter the following command.

```
ServerIronADX(config)# show ip acl-traffic

ICMP inbound packets received 400
ICMP inbound packets permitted 200
ICMP inbound packets denied 200
```

Syntax: show ip acl-traffic

The command lists a separate set of statistics for each of the following IP protocols:

- ICMP

- IGMP
- IGRP
- IP
- OSPF
- TCP
- UDP
- Protocol number, if an ACL is configured for a protocol not listed above

For TCP and UDP, a separate set of statistics is listed for each application port.

Clearing flow-based ACL statistics

To clear the ACL statistics, enter the following command at the Privileged EXEC level of the CLI.

```
ServerIronADX(config)# clear ip acl-traffic
```

Syntax: clear ip acl-traffic

Dropping all fragments that exactly match a flow-based ACL

For a packet fragment that is sent to the CPU for processing, the device compares the fragment's source and destination IP addresses against the interface's ACL entries. By default, if the fragment's source and destination IP addresses exactly match an ACL entry that also has Layer 4 information (source and destination TCP or UDP application ports), the device permits or denies the fragment according to the ACL.

On an individual interface basis, you can configure an IronCore device to automatically drop a fragment whose source and destination IP addresses exactly match an ACL entry that has Layer 4 information, even if that ACL entry's action is permit. To do so, enter the following command at the configuration level for an interface.

```
ServerIronADX(config-if-1/1)# ip access-group frag deny
```

Syntax: [**no**] **ip access-group frag deny**

Clearing the ACL statistics

Statistics on the ACL account report can be cleared:

- When a software reload occurs
- When the ACL is bound to or unbound from an interface
- When you enter the **clear access-list** command, as in the following example.

```
ServerIronADX(config)# clear access-list all
```

Syntax: clear access-list all | **ethernet** <*slot*>/<*port*>

Enter **all** to clear all statistics for all ACLs.

Use **ethernet** <*slot*>/<*port*> to clear statistics for ACLs a physical port.

Enabling ACL filtering of fragmented packets

This section includes the following topics:

- "Filtering fragmented packets for rule-based ACLs" on page 86
- "Throttling the fragment rate" on page 86

Filtering fragmented packets for rule-based ACLs

By default, when a rule-based ACL is applied to a port, the port will use the ACL to permit or deny the first fragment of a fragmented packet, but forward subsequent fragments of the same packet in hardware. Generally, denying the first fragment of a packet is sufficient, since a transaction cannot be completed without the entire packet.

NOTE
The fragmentation support described in this section applies only to rule-based ACLs.

NOTE
Enhanced fragment handling is not supported on 10 Gigabit Ethernet modules. By default, 10 Gigabit Ethernet modules also forward the first fragment instead of using the ACLs to permit or deny the fragment.

For tighter control, you can enable CPU filtering of all packet fragments on a port. When you enable CPU filtering, the port sends all the fragments of a fragmented packet to the CPU. The CPU then permits or denies each fragment according to the ACL applied to the port. You can enable CPU filtering of fragments on individual ports.

You also can configure the port to drop all packet fragments.

To enable CPU filtering of packet fragments on an individual port, enter commands such as the following.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip access-group frag inspect
```

Syntax: [no] **ip access-group frag inspect** | **deny**

The **inspect** | **deny** parameter specifies whether you want fragments to be sent to the CPU or dropped:

- **inspect**: This option sends all fragments to the CPU.
- **deny**: This option begins dropping all fragments received by the port as soon as you enter the command. This option is especially useful if the port is receiving an unusually high rate of fragments, which can indicate a hacker attack.

Throttling the fragment rate

By default, when you enable CPU filtering of packet fragments, all fragments are sent to the CPU. Normally, the fragment rate in a typical network does not place enough additional load on the CPU to adversely affect performance. However, performance can be affected if the device receives a very high rate of fragments. For example, a misconfigured server or a hacker can affect the device's performance by flooding the CPU with fragments.

You can protect against fragment flooding by specifying the maximum number of fragments the device or an individual interface is allowed to send to the CPU in a one-second interval. If the device or an interface receives more than the specified number of fragments in a one-second interval, the device either drops or forwards subsequent fragments in hardware, depending on the action you specify. In addition, the device starts a holddown timer and continues to either drop or forward fragments until the holddown time expires.

The device also generates a syslog message.

To specify the maximum fragment rate per second, enter commands such as the following.

```
ServerIronADX(config)# ip access-list frag-rate-on-sys 15000 exceed-action deny
reset-interval 10
ServerIronADX(config)#ip access-list frag-rate-on-interface 5000 exceed-action
forward reset-interval 5
```

The first command sets the fragment threshold at 15,000 per second, for the entire device. If the device receives more than 15,000 packet fragments in a one-second interval, the device takes the specified action. The action specified with this command is to drop the excess fragments and continue dropping fragments for a holddown time of ten minutes. After the ten minutes have passed, the device starts sending fragments to the CPU again for processing.

The second command sets the fragment threshold at 5,000 for individual interfaces. If any interface on the device receives more than 5,000 fragments in a one-second interval, the device takes the specified action. In this case, the action is to forward the fragments in hardware without filtering them. The device continues forwarding fragments in hardware for five minutes before beginning to send fragments to the CPU again.

Both thresholds apply to the entire device. Thus, if an individual interface's fragment threshold is exceeded, the drop or forward action and the holddown time apply to all fragments received by the device.

Syntax: [**no**] **ip access-list frag-rate-on-sys** <*num*> **exceed-action deny** | **forward reset-interval** <*mins*>

and

Syntax: [**no**] **ip access-list frag-rate-on-interface** <*num*> **exceed-action drop** | **forward reset-interval** <*mins*>

The <*num*> parameter specifies the maximum number of fragments the device or an individual interface can receive and send to the CPU in a one-second interval.

- **frag-rate-on-sys:** Sets the threshold for the entire device. The device can send to the CPU only the number of fragments you specify per second, regardless of which interfaces the fragments come in on. If the threshold is exceeded, the device takes the exceed action you specify.
- **frag-rate-on-interface**: Sets the threshold for individual interfaces. If an individual interface receives more than the specified maximum number of fragments, the device takes the exceed action you specify.

The <*num*> parameter specifies the maximum number of fragments per second.

- For **frag-rate-on-system**, you can specify from 600 through 12800. The default is 6400.
- For **frag-rate-on-interface**, you can specify from 300 through 8000. The default is 4000.

The **drop | forward** parameter specifies the action to take if the threshold (<*num*> parameter) is exceeded:

- **drop**: Fragments are dropped without filtering by the ACLs

- **forward**: Fragments are forwarded in hardware without filtering by the ACLs

The <*mins*> parameter specifies the number of minutes the device will enforce the drop or forward action after a threshold has been exceeded. You can specify from 1 to 30 minutes, for **frag-rate-on-sys** or **frag-rate-on-interface**.

Syslog messages for exceeded fragment thresholds

If a fragment threshold is exceeded, the device generates one of the syslog messages shown in Table 11.

TABLE 11 Syslog messages for exceeded fragment threshold

Message level	Message	Explanation
Notification	ACL system fragment packet inspect rate <*rate*> exceeded	The <*rate*> indicates the maximum rate allowed.
Notification	ACL port fragment packet inspect rate <*rate*> exceeded on port <*portnum*>	The <*rate*> indicates the maximum rate allowed. The <*portnum*> indicates the port.

Enabling filtering for packets denied by flow-based ACLs

By default, packets denied by ACLs are filtered by the CPU. You can enable the device to create CAM entries for packets denied by ACLs. This causes the filtering to occur in hardware instead of in the CPU.

When you enable hardware filtering of denied packets, the first time the device filters a packet denied by an ACL, the device sends the packet to the CPU for processing. The CPU also creates a CAM entry for the denied packet. Subsequent packets with the same address information are filtered using the CAM entry. The CAM entry ages out after two minutes if not used.

To enable hardware filtering of denied packets, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# hw-drop-acl-denied-packet
```

Syntax: [**no**] **hw-drop-acl-denied-packet**

Enabling strict TCP or UDP mode for flow-based ACLs

By default, when you use ACLs to filter TCP or UDP traffic, the Foundry device does not compare all TCP or UDP packets against the ACLs.

For TCP and UDP, the device first compares the source and destination information in a TCP control packet or a UDP packet against entries in the session table. The session table contains forwarding entries based on Layer 3 and Layer 4 information:

- If the session table contains a matching entry, the device forwards the packet, assuming that the first packet the device received with the same address information was permitted by the ACLs.

- If the session table does not contain a matching entry, the device sends the packet to the CPU, where the software compares the packet against the ACLs. If the ACLs permit the packet (explicitly by a permit ACL entry or implicitly by the absence of a deny ACL entry), the CPU creates a session table entry for the packet's forwarding information and forwards the packet.

For TCP, this behavior by default applies only to control packets, not to data packets. Control packets include packet types such as SYN (Synchronization) packets, FIN (Finish) packets, and RST (Reset) packets.

For tighter access or forwarding control, you can enable the device to perform strict TCP or UDP ACL processing. The following sections describe the strict modes in more detail.

Enabling strict TCP mode

By default, when you use ACLs to filter TCP traffic, the Foundry device does not compare all TCP packets against the ACLs. Instead, the device compares TCP control packets against the ACLs, but not data packets. Control packets include packet types such as SYN (Synchronization) packets, FIN (Finish) packets, and RST (Reset) packets.

In normal TCP operation, TCP data packets are present only if a TCP control session for the packets also is established. For example, data packets for a session never occur if the TCP SYN for that session is dropped. Therefore, by filtering the control packets, the Foundry device also implicitly filters the data packets associated with the control packets. This mode of filtering optimizes forwarding performance for TCP traffic by forwarding data packets without examining them. Since the data packets are present in normal TCP traffic only if a corresponding TCP control session is established, comparing the packets for the control session to the ACLs is sufficient for filtering the entire session including the data.

However, it is possible to generate TCP data packets without corresponding control packets, in test or research situations for example. In this case, the default ACL mode does not filter the data packets, since there is no corresponding control session to filter. To filter this type of TCP traffic, use the strict ACL TCP mode. This mode compares all TCP packets to the configured ACLs, regardless of whether the packets are control packets or data packets. If the ACLs permit the packet, the device creates a session entry for forwarding other TCP packets with the same Layer 3 and Layer 4 addresses.

NOTE
Regardless of whether the strict mode is enabled or disabled, the device always compares TCP control packets against the configured ACLs before creating a session entry for forwarding the traffic.

NOTE
If the device's configuration currently has ACLs associated with interfaces, remove the ACLs from the interfaces before changing the ACL mode.

To enable the strict ACL TCP mode, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip strict-acl-tcp
```

Syntax: [no] ip strict-acl-tcp

This command configures the device to compare all TCP packets against the configured ACLs before forwarding them.

To disable the strict ACL mode and return to the default ACL behavior, enter the following command.

```
ServerIronADX(config)# no ip strict-acl-tcp
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-tcp** or **no ip strict-acl-tcp** command into effect.

Enabling strict UDP mode

By default, when you use ACLs to filter UDP traffic, the Foundry device does not compare all UDP packets against the ACLs. Instead, the device compares the source and destination information against entries in the session table. The session table contains forwarding entries based on Layer 3 and Layer 4 information:

- If the session table contains a matching entry, the device forwards the packet, assuming that the first packet the device received that contains the same address information was permitted by the ACLs.
- If the session table does not contain a matching entry, the device sends the packet to the CPU, where the software compares the packet against the ACLs. If the ACLs permit the packet (explicitly by a permit ACL entry or implicitly by the absence of a deny ACL entry), the CPU creates a session table entry for the packet's forwarding information and forwards the packet.

For tighter control, the software provides the strict ACL UDP mode. When you enable strict UDP processing, the device sends every UDP packet to the CPU and compares the packet against the configured ACLs.

NOTE
If the device's configuration currently has ACLs associated with interfaces, remove the ACLs from the interfaces before changing the ACL mode.

To enable the strict ACL UDP mode, enter the following command at the global CONFIG level of the CLI.

```
ServerIronADX(config)# ip strict-acl-udp
```

Syntax: [no] **ip strict-acl-udp**

This command configures the device to compare all UDP packets against the configured ACLs before forwarding them.

To disable the strict ACL mode and return to the default ACL behavior, enter the following command.

```
ServerIronADX(config)# no ip strict-acl-udp
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-udp** or **no ip strict-acl-udp** command into effect.

Configuring ACL packet and flow counters

You can configure counters for packets and flows that match entries in an ACL. Using the CLI, you can display the contents of the counters and clear them:

- The ACL packet counter feature provides an accurate count of packets matching individual ACL entries.
- The ACL flow counter feature provides an approximate count of flows matching individual ACL entries. This feature can be used for troubleshooting purposes to provide an indication of flow activity against an ACL. Each time the Foundry device receives the first packet of a flow matching an entry in an ACL list, the flow counter for that ACL entry is incremented by one. If a flow lasts longer than two minutes, the flow counter for the ACL entry is incremented again.

NOTE
The ACL flow counter feature is designed to monitor the general volume of flow activity for an ACL. It is not intended to be used for accounting purposes.

The ACL flow and packet counters are incremented differently depending on whether packets are handled by the Management Processor (MP), and whether they are permit or deny flows.

The Management Processor (MP) handles flows as follows.

For flows handled by the Management Processor:

- For permit flows, only flows are counted. If a permit flow lasts longer than two minutes, the flow counter is incremented again.
- For deny flows, only packets are counted.

Once the ACL packet and flow counters are enabled, you can disable them with the **no** form of the **enable-acl-counter** command. Disabling and then re-enabling the ACL packet and flow counters resets them to zero.

To display the packet and flow counters for ACL 100.

```
ServerIronADX# show access-list 100
Extended IP access list 100 (Total flows: 432, Total packets: 42000)
  permit tcp 1.1.1.0 0.0.0.255 any (Flows: 80, Packets: 12900)
  deny udp 1.1.1.0 0.0.0.255 any (Flows: 121, Packets: 20100)
  permit ip 2.2.2.0 0.0.0.255 any (Flows: 231, Packets: 9000)
```

Syntax: show access-list <*acl-num*> | <*acl-name*> | **all**

To clear the flow counters for ACL 100.

```
ServerIronADX# clear access-list 100
```

Syntax: clear access-list <*acl-num*> | <*acl-name*> | **all**

ACLs and ICMP

This section describes how ACLs can be used to filter traffic based on ICMP packets.

Using flow-based ACLs to filter ICMP packets

To configure an extended ACL that filters based on the IP packet length of ICMP packets, enter commands such as the following.

```
ServerIronADX(config)#access-list 105 deny icmp any any echo ip-pkt-len 92
ServerIronADX(config)#access-list 105 deny icmp any any echo ip-pkt-len 100
ServerIronADX(config)#access-list 105 permit ip any any
```

The commands in this example deny (drop) ICMP echo request packets that contain a total length of 92 or 100 in the IP header field. You can specify an IP packet length of 1 through 65535. Refer to the section "ICMP filtering with flow-based ACLs" on page 92 for additional information on using ICMP to filter packets.

ICMP filtering with flow-based ACLs

Most Foundry software releases that support flow-based ACLs filter traffic based on the following ICMP message types:

- echo
- echo-reply
- information-request
- mask-reply
- mask-request
- parameter-problem
- redirect
- source-quench
- time-exceeded
- timestamp-reply
- timestamp-request
- unreachable
- *<num>*

Also, to create ACL policies that filter ICMP message types, you can either enter the description of the message type or enter its type and code IDs. Furthermore ICMP message type filtering is now available for rule-based ACLs on BigIron Layer 2 Switch and Layer 3 Switch images.

Numbered ACLs

For example, to deny the echo message type in a numbered ACL, enter commands such as the following when configuring a numbered ACL.

```
ServerIronADX(config)# access-list 109 deny ICMP any any echo
```

or

```
ServerIronADX(config)# access-list 109 deny ICMP any any 8 0
```

Syntax: [**no**] **access-list** <*num*>

Syntax: **deny** | **permit icmp** <*source-ip-address*> **|** <*source-ip-address/subnet-mask*> | **any** | **host** <*source-host*>
<*destination-ip-address*> | <*destination-ip-address/subnet-mask*> | **any** | **host** <*destination-host*>
<*icmp-type*> | <*icmp-type-number*> <*icmp-code-number*>

The **deny** | **permit** parameter indicates whether packets that match the policy are dropped or forwarded.

You can either enter the name of the message type for <*icmp-type*> or the type number and code number of the message type. Refer to Table 12 on page 93 for valid values.

Named ACLs

For example, to deny the administratively-prohibited message type in a named ACL, enter commands such as the following.

```
ServerIronADX(config)# ip access-list extended melon
ServerIronADX(config-ext-nacl)# deny ICMP any any
```

or

```
ServerIronADX(config)# ip access-list extended melon
ServerIronADX(config-ext-nacl)# deny ICMP any any 3 13
```

Syntax: [**no**] **ip access-list extended** <*acl-num*> | <*acl-name*>

Syntax: **deny** | **permit icmp** <*source-ip-address*> | <*source-ip-address/subnet-mask*> | **any** | **host** <*source-host*>
<*destination-ip-address*> | **destination-ip-address/subnet-mask>** | **any** | **host** <*destination-host*>
<*icmp-type*> | <*icmp-type-number*> <*icmp-code-number*>

The **extended** parameter indicates the ACL entry is an extended ACL.

The <*acl-name*> | <*acl-num*> parameter allows you to specify an ACL name or number. If using a name, specify a string of up to 256 alphanumeric characters. You can use blanks in the ACL name if you enclose the name in quotation marks (for example, “ACL for Net1”). The <*acl-num*> parameter allows you to specify an ACL number if you prefer. If you specify a number, enter a number from 100 through 199 for extended ACLs.

The **deny** | **permit** parameter indicates whether packets that match the policy are dropped or forwarded.

You can either use the <*icmp-type*> and enter the name of the message type or use the <*icmp-type-number*> <*icmp-ode-number*> parameter and enter the type number and code number of the message. Refer to Table 12 for valid values.

NOTE
“X” in the Type-Number or Code-Number column in Table 12 means the device filters any traffic of that ICMP message type.

TABLE 12 ICMP message types and codes

ICMP message type	Type	Code
administratively-prohibited	3	13
any-icmp-type	x	x
destination-host-prohibited	3	10
destination-host-unknown	3	7
destination-net-prohibited	3	9
destination-network-unknown	3	6
echo	8	0
echo-reply	0	0
general-parameter-problem	12	1

NOTE: This message type indicates that required option is missing.

TABLE 12 ICMP message types and codes (Continued)

ICMP message type	Type	Code
host-precedence-violation	3	14
host-redirect	5	1
host-tos-redirect	5	3
host-tos-unreachable	3	12
host-unreachable	3	1
information-request	15	0
log		
mask-reply	18	0
mask-request	17	0
net-redirect	5	0
net-tos-redirect	5	2
net-tos-unreachable	3	11
net-unreachable	3	0
packet-too-big	3	4
parameter-problem **NOTE:** This message includes all parameter problems	12	0
port-unreachable	3	3
precedence-cutoff	3	15
protocol-unreachable	3	2
reassembly-timeout	11	1
redirect **NOTE:** This includes all redirects.	5	x
router-advertisement	9	0
router-solicitation	10	0
source-host-isolated	3	8
source-quench	4	0
source-route-failed	3	5
time-exceeded	11	x
timestamp-reply	14	0
timestamp-request	13	0
ttl-exceeded	11	0
unreachable **NOTE:** This includes all unreachable messages	3	x

Using flow-based ACLs and NAT on the same interface

You can use ACLs and NAT on the same interface, as long as you follow these guidelines:

- You must use the **ip strict-acl-tcp** command when configuring ACLs and NAT is configured on the same Layer 2 Switch. (Refer to the instructions below on how to use this command.)
- Do not enable NAT on an interface until you have applied ACLs (as described below) to the interface. If NAT is already enabled, you must disable it, apply the ACLs, then re-enable NAT on the interface.
- Enable the strict TCP mode.
- On the inside NAT interface (the one connected to the private addresses), apply inbound ACLs that permit TCP, UDP, and ICMP traffic to enter the device from the private subnet.

You can use a standard ACL to permit all traffic (including TCP, UDP, and ICMP traffic) or an extended ACL with separate entries to explicitly permit TCP, UDP, and ICMP traffic.

NOTE
You do not need to apply ACLs to permit TCP, UDP, and ICMP traffic unless you are applying other ACLs to the interface as well. If you do not plan to apply any ACLs to a NAT interface, then you do not need to apply the ACLs to permit TCP, UDP, and ICMP traffic.

Here is an example of how to configure device to use ACLs and NAT on the same interfaces. In this example, the inside NAT interface is port 1/1 and the outside NAT interface is port 2/2.

The following commands enable the strict TCP mode and configure an ACL to permit all traffic from the 10.10.200.x subnet. A second ACL denies traffic from a specific host on the Internet.

```
ServerIronADX(config)# ip strict-acl-tcp
ServerIronADX(config)# access-list 1 permit 10.10.200.0 0.0.0.255
ServerIronADX(config)# access-list 2 deny 209.157.2.184
```

The following commands configure global NAT parameters.

```
ServerIronADX(config)# ip nat inside source list 1 pool outadds overload
ServerIronADX(config)# ip nat pool outadds 204.168.2.1 204.168.2.254 netmask
255.255.255.0
```

The following commands configure the inside and outside NAT interfaces. Notice that the ACLs are applied to the inbound direction on the inside NAT interface, and are applied *before* NAT is enabled. In this example, ACL 1 permits all traffic to come into the inside interface from the private subnet. ACL 2 denies traffic from a specific host from going out the interface to the private subnet.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ip address 10.10.200.1 255.255.255.0
ServerIronADX(config-if-1/1)# ip access-group 1 in
ServerIronADX(config-if-1/1)# ip nat inside
ServerIronADX(config-if-1/1)# interface ethernet 2/2
ServerIronADX(config-if-2/2)# ip address 204.168.2.78 255.255.255.0
ServerIronADX(config-if-2/2)# ip nat outside
```

NOTE
Enter the **ip rebind-acl** command at the global CONFIG level of the CLI to place the **ip strict-acl-tcp** command into effect.

Troubleshooting rule-based ACLs

Use the following methods to troubleshoot a rule-based ACL:

- To display the number of Layer 4 CAM entries being used by each ACL, enter the **show access-list all** command. Refer to "Displaying the number of Layer 4 CAM entries" on page 64.
- To view the types of packets being received on an interface, enable ACL statistics using the **enable-acl-counter** command, reapply the ACLs by entering the **ip rebind-acl all** command, then display the statistics by entering the **show ip acl-traffic** command.
- To determine whether an ACL entry is correctly matching packets, add the **log** option to the ACL entry, then reapply the ACL. This forces the device to send packets that match the ACL entry to the CPU for processing. The **log** option also generates a syslog entry for packets that are permitted or denied by the ACL entry.
- To determine whether the issue is specific to fragmentation, remove the Layer 4 information (TCP or UDP application ports) from the ACL, then reapply the ACL.

If you are using another feature that requires ACLs, either use the same ACL entries for filtering and for the other feature, or change to flow-based ACLs.

Chapter

IPv6 Access Control Lists

6

In this chapter

- In this chapter 97
- Configuring IPv6 ACLs 99
- Applying IPv6 ACLs to interfaces 105
- Displaying IPv6 ACLs 106
- Using an ACL to restrict SSH access 107
- Using an ACL to restrict Telnet access 107
- Logging IPv6 ACLs 107

IPv6 ACL overview

ServerIron ADX supports IPv6 access control lists (ACLs) in hardware. You can configure up to a maximum of 1024 ACL entries in any combination of different ACLs. The total number of entries in all ACLs cannot exceed the system maximum of 1024 entries.

By default, IPv6 ACLs are processed in hardware and all IPv6 ACL rules are stored in ternary content-addressable memory (TCAM).

An IPv6 ACL is composed of one or more conditional statements that cause an action (permit or deny) if a packet matches a specified source or destination prefix. If the maximum number of IPv6 ACL rules is reached, the following error message is displayed on the console:

```
IPv6 Hardware ACL rules cannot be configured,exceeds the maximum hardware limit of
1024 entries
Insufficient hardware resource for binding the ACL scale1 to interface Port or
Slot/Port.
```

In ACLs with multiple statements, you can specify a priority for each statement. The specified priority determines the order in which the statement appears in the ACL. The last statement in each IPv6 ACL is an implicit deny statement for all packets that do not match the previous statements in the ACL.

You can configure an IPv6 ACL on a global basis, and then apply it to the incoming IPv6 packets on specified interfaces. You can apply only one IPv6 ACL to an interface's incoming traffic. When an interface receives an IPv6 packet, it applies the statements within the ACL in their order of appearance to the packet. As soon as a match occurs, the ServerIron ADX takes the specified action (permit or deny the packet) and stops further comparison for that packet.

Foundry IPv6 ACLs enable traffic filtering based on the following information:

- IPv6 protocol
- Source IPv6 address
- Destination IPv6 address

- Source TCP or UDP port (if the IPv6 protocol is TCP or UDP)
- Destination TCP or UDP port (if the IPv6 protocol is TCP or UDP)

The IPv6 protocol can be one of the following well-known names or any IPv6 protocol number from 0 through 255:

- IP Authentication Header (AH)
- Encapsulating Security Payload (ESP)
- Internet Control Message Protocol (ICMP)
- Internet Protocol version 6 (IPv6)
- Stream Control Transmission Protocol (SCTP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

NOTE
TCP and UDP filters will be matched only if they are listed as the first option in the extension header.

For TCP and UDP, you can specify a comparison operator and port name or number. For example, you can configure a policy to block web access to a specific website by denying all TCP port 80 (HTTP) packets from a specified source IPv6 address to the website's IPv6 address.

Configuration notes

- Either IPv6 must be enabled globally or an IPv6 address must be configured on an interface before IPv6 ACLs can be configured.
- An IPv6 ACL can include up to 1024 entries or statements.
- Only named ACLs are supported.
- Only inbound ACLs are supported.
- If an IPv6 ACL has the implicit deny condition, make sure it also permits the IPv6 link-local address in addition to the global unicast address. Otherwise, routing protocols such as OSPF will not work. To view the link-local address, use the **show ipv6 interface** command.
- You cannot disable IPv6 on an interface to which an ACL is bound. Attempting to do so will cause the system to return the following error message.

```
Error:  Port 7 has IPv6 ACL configured.  Cannot disable IPv6
```

To disable IPv6, first remove the ACL from the interface.

Processing of IPv6 ACLs

There are two ways that IPv6 ACLs are processed in Foundry devices: in software and in hardware. The processing differs depending on the software release that you are running. These differences are described in the following sections.

Prior to ServerIron ADX 12.3.01

Prior to the release of ServerIron ADX 12.3.01, all permit and deny packets for IPv6 ACLs are forwarded to the barrel processors (BPs) and the BPs perform the ACL processing.

Beginning with ServerIron ADX 12.3.01 and later

Beginning with release 12.3.01, IPv6 ACLs are processed as described for the following actions.

For deny actions

All deny packets are dropped in hardware.

For permit actions

For all permit traffic, packets are processed in hardware and then forwarded to the barrel processors (BPs). The BPs do not take any action on the ACLs.

Backward compatibility option

You can use the **ipv6 flow-based-acl-enable** command to provide backward compatibility for IPv6 ACL processing. If this command is configured, packets are processed in hardware and then forwarded to the BPs where the BPs also process the ACLs.

```
ServerIronADX(config)# ipv6 flow-based-acl-enable
```

Syntax: ipv6 flow-based-acl-enable

Configuring IPv6 ACLs

To configure an IPv6 ACL, complete the following steps:

1. Create the IPv6 ACL.
2. Apply the IPv6 ACL to the interface.

Example configurations

To configure an ACL that blocks all Telnet traffic received on port 1/1 from IPv6 host 2001:db8:2382:e0bb::2, enter the following commands.

```
ServerIronADX(config)# ipv6 access-list fdry
ServerIronADX(config-ipv6-access-list-fdry)# deny tcp host 2001:db8:2382:e0bb:
:2 any eq telnet
ServerIronADX(config-ipv6-access-list-fdry)# permit ipv6 any any
ServerIronADX(config-ipv6-access-list-fdry)# exit
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-1/1)# ipv6 traffic-filter fdry in
ServerIronADX(config)# write memory
```

Here is another example of commands for configuring an ACL and applying it to an interface.

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp 2001:db8:2383:
e0bb::/64 2001:db8:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# deny ipv6 host 2001:db8:2383:
e0ac::2 host 2001:db8:2383:e0aa:0::24
ServerIronADX(config-ipv6-access-list-netw)# deny udp any any
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

The first condition permits ICMP traffic from hosts in the 2001:db8:2383:e0bb::/64 network to hosts in the 2001:db8:3782::/64 network.

The second condition denies all IPv6 traffic from host 2001:db8:2383:e0ac::2 to host 2001:db8:2383:e0aa:0::24.

The third condition denies all UDP traffic.

The fourth condition permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL would deny all incoming IPv6 traffic on the ports to which you assigned the ACL.

The following commands apply the ACL named "netw" to the incoming traffic on port 1/2 and to the incoming traffic on port 4/3.

```
ServerIronADX(config)# interface ethernet 1/2
ServerIronADX(config-if-1/2)# ipv6 traffic-filter netw in
ServerIronADX(config-if-1/2)# exit
ServerIronADX(config)# interface ethernet 4/3
ServerIronADX(config-if-4/3)# ipv6 traffic-filter netw in
ServerIronADX(config)# write memory
```

The folloiwng example swhos the commands for configuring another ACL and applying it to an interface.

```
ServerIronADX(config)# ipv6 access-list nextone
ServerIronADX(config-ipv6-access-list rtr)# deny tcp 2001:db8:1570:21::/24
2001:db8:1570:22::/24
ServerIronADX(config-ipv6-access-list rtr)# deny udp any range 5 6
2001:db8:1570:22::/24
ServerIronADX(config-ipv6-access-list rtr)# permit ipv6 any any
ServerIronADX(config-ipv6-access-list rtr)# write memory
```

The first condition in this ACL denies TCP traffic from the 2001:db8:1570:21::/24 network to the 2001:db8:1570:22::/24 network.

The next condition denies UDP packets from source UDP port 5 or 6 to the 2001:db8:1570:22::/24 network.

The third condition permits all packets containing source and destination addresses that are not explicitly denied by the first two conditions. Without this entry, the ACL would deny all incoming IPv6 traffic on the ports to which you assign the ACL.

The **show running-config** command displays the following output:

```
ServerIronADX(config)# show running-config
ipv6 access-list rtr
deny tcp 2001:db8:1570:21::/24 2001:db8:1570:22::/24
deny udp any range 5 6 2001:db8:1570:22::/24
permit ipv6 any any
```

The **show ipv6 access-list** command displays the following output:

```
ServerIronADX(config)# show ipv6 access-list rtr
ipv6 access-list rtr: 3 entries
deny tcp 2001:db8:1570:21::/24 2001:db8:1570:22::/24
deny udp any range 5 6 2001:db8:1570:22::/24
permit ipv6 any any
```

The following commands apply the ACL called “rtr” to the incoming traffic on ports 2/1 and 2/2.

```
ServerIronADX(config)# interface ethernet 2/1
ServerIronADX(config-if-2/1)# ipv6 traffic-filter rtr in
ServerIronADX(config-if-2/1)# exit
ServerIronADX(config)# interface ethernet 2/2
ServerIronADX(config-if-2/2)# ipv6 traffic-filter rtr in
ServerIronADX(config)# write memory
```

Default and implicit IPv6 ACL actions

When no IPv6 ACLs are configured on an interface, all IPv6 traffic is permitted by default. Once you configure an IPv6 ACL and apply it to an interface, all IPv6 traffic no explicitely permitted is denied by default.

- If you want to control access, configure ACLs consisting of permit entries for the access you want to permit. The ACLs implicitly deny all other access.
- If you want to secure access in environments with many users, you may want to configure ACLs that consist of explicit deny entries, and then add an entry to the end of each ACL to permit all access. The permit entry allows packets that are not explicitly denied by the deny entries.

Every IPv6 ACL uses the the **deny ipv6 any any** implicit condition as its last match condition to deny IPv6 traffic. You must enter a **permit ipv6 any any** condition as the last statement in the ACL if you want to permit IPv6 traffic that were not denied by the previous statements.

NOTE
If an IPv6 ACL has the implicit deny condition, make sure it also permits the IPv6 link-local address in addition to the global unicast address. Otherwise, routing protocols such as OSPF will not work. To view the link-local address, use the **show ipv6 interface** command.

The conditions are applied in the order shown above, with **deny ipv6 any any** as the last condition applied.

For example, if you want to deny ICMP neighbor discovery acknowledgement, and permit any remaining IPv6 traffic, enter commands such as in the following example.

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp 2001:db8:2383:e0bb:
:/64 2001:db8:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

The first permit statement permits ICMP traffic from hosts in the 2001:db8:2383:e0bb::/64 network to hosts in the 2001:db8:3782::/64 network.

The last entry permits all packets that are not explicitly denied by the other entries. Without this entry, the ACL will deny all incoming IPv6 traffic on the ports to which you assigned the ACL.

If you add the statement **deny icmp any any** to the ACL all neighbor discovery messages will be denied.

```
ServerIronADX(config)# ipv6 access-list netw
ServerIronADX(config-ipv6-access-list-netw)# permit icmp
2001:db8:2383:e0bb:
:/64 2001:db8:3782::/64
ServerIronADX(config-ipv6-access-list-netw)# deny icmp any any
ServerIronADX(config-ipv6-access-list-netw)# permit ipv6 any any
```

The deny statement denies ICMP neighbor discovery acknowledgement.

IPv6 ACL syntax

When creating IPv6 ACLs, you must use the syntax that is appropriate to the protocol you are filtering. The following sections show the IPv6 ACL syntax for the ICMP, TCP, UDP, and other supported protocols.

ICMP syntax

Syntax: [**no**] **ipv6 access-list** <*acl-name*>

Syntax: **permit** | **deny icmp** <*ipv6-source-prefix/prefix-length*> | **any** | **host** <*source-ipv6_address*>
<*ipv6-destination-prefix/prefix-length*> | **any** | **host** <*ipv6-destination-address*>
[**ipv6-operator** [<*value*>]]
[[<*icmp-type*>][<*icmp-code*>]] | [<*icmp-message*>] [**log**]

TCP syntax

Syntax: [**no**] **ipv6 access-list** <*acl-name*>

Syntax: **permit | deny tcp**
<*ipv6-source-prefix/prefix-length*> | **any** | **host** <*source-ipv6_address*> [**tcp-udp-operator** <*source-port-number*>]
<*ipv6-destination-prefix/prefix-length*> | **any** | **host** <*ipv6-destination-address*>
[**tcp-udp-operator <***destination-port-number*>]
[**ipv6-operator** [<*value*>]] [**log**]

UDP syntax

Syntax: [**no**] **ipv6 access-list** <*acl-name*>

Syntax: **permit** | **deny udp**
<*ipv6-source-prefix/prefix-length*> | **any** | **host** <*source-ipv6_address*> [**tcp-udp-operator** <*source-port-number*>]
<*ipv6-destination-prefix/prefix-length*> | **any** | **host** <*ipv6-destination-address*>
[**tcp-udp-operator** <*destination-port-number*>]
[**ipv6-operator** [<*value*>]] [**log**]

IPv6 and supported protocols other than ICMP, TCP, or UDP

Syntax: [**no**] i**pv6 access-list** <*acl-name*>

Syntax: **permit** | **deny** <*protocol*>
<*ipv6-source-prefix/prefix-length*> | **any** | **host** <*source-ipv6_address*>
<*ipv6-destination-prefix/prefix-length*> | **any** | **host** <*ipv6-destination-address*>
[**ipv6-operator** [<*value*>]] [**log**]

Table 13 describes the syntax used to configure IPv6 ACLs.

TABLE 13 Syntax descriptions

Syntax	Description
ipv6 access-list <*acl-name*>	Enables the IPv6 configuration level and defines the name of the IPv6 ACL. The <*acl-name*> can contain up to 199 characters and numbers, but cannot begin with a number and cannot contain any spaces or quotation marks.
permit	The ACL will permit (forward) packets that match a policy in the ACL.
deny	The ACL will deny (drop) packets that match a policy in the ACL.
icmp	Indicates the you are filtering ICMP packets.
<*protocol*>	The type of IPv6 packet you are filtering. You can specify a well-known name for a protocol with a number less than 255. For other protocols, you must enter the number. Enter "?" instead of a protocol to list the well-known names recognized by the CLI. IPv6 protocols include: AHP: Authentication Header ESP: Encapsulating Security Payload ICMP: Internet Protocol Message Protocol IPv6: Internet Protocol version 6 SCTP: Stream Control Transmission Protocol TCP: Transmission Control Protocol UDP: User Datagram Protocol
<*ipv6-source-prefix/prefix-length*>	The <*ipv6-source-prefix/prefix-length*> variable specifies a source prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <*ipv6-source-prefix*> variable in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <*prefix-length*> variable as a decimal value. A slash mark (/) must follow the <*ipv6-source-prefix*> variable and precede the <*prefix-length*> variable.
<*ipv6-destination-prefix/prefix-length*>	The <*ipv6-destination-prefix/prefix-length*> variable specifies a destination prefix and prefix length that a packet must match for the specified action (deny or permit) to occur. You must specify the <*ipv6-destination-prefix*> variable in hexadecimal using 16-bit values between colons as documented in RFC 2373. You must specify the <*prefix-length*> variable as a decimal value. A slash mark (/) must follow the <*ipv6-destination-prefix*> variable and precede the <*prefix-length*> variable.
<*source-ipv6-address*>	
<*ipv6-destination-address*>	
any	When specified instead of the <*ipv6-source-prefix*>/<*prefix-length*> or <*ipv6-destination-prefix*>/<*prefix-length*> variables, matches any IPv6 prefix and is equivalent to the IPv6 prefix::/0.
host	Allows you specify a host IPv6 address. When you use this parameter, you do not need to specify the prefix length. A prefix length of all 128 is implied.
<*icmp-message***>**	ICMP packets are filtered by ICMP messages. See the "Configuring IPv6 ICMP Features" section of the "Configuring IPv6 Connectivity" chapter of the *ServerIron ADX TrafficWorks Switching and Routing Guide*.

TABLE 13 Syntax descriptions (Continued)

Syntax	Description
tcp	Indicates the you are filtering TCP packets.
udp	Indicates the you are filtering UDP packets.
tcp-udp-operator <*source-port-number*> |<*destination-port-number*>	The **tcp-udp-operator** parameter can be one of the following: • **eq**: The policy applies to the TCP or UDP port name or number you enter after **eq**. • **gt**: The policy applies to TCP or UDP port numbers greater than the port number or the numeric equivalent of the port name you enter after **gt**. Enter "?" to list the port names. • **lt**: The policy applies to TCP or UDP port numbers that are less than the port number or the numeric equivalent of the port name you enter after **lt**. • **neq**: The policy applies to all TCP or UDP port numbers except the port number or port name you enter after **neq**. • **range**: The policy applies to all TCP or UDP port numbers that are between the first TCP or UDP port name or number and the second one you enter following the **range** parameter. The range includes the port names or numbers you enter. For example, to apply the policy to all ports between and including 23 (Telnet) and 53 (DNS), enter the following: **range 23 53**. The first port number in the range must be lower than the last number in the range. The <*source-port-number*> and <*destination-port-number*> for the **tcp-udp-operator** is the number of the port.
ipv6-operator <*value*>	Allows you to filter the packets further by using one of the following options: • **dscp**: The policy applies to packets that match the traffic class value in the traffic class field of the IPv6 packet header. This operator allows you to filter traffic based on TOS or IP precedence. You can specify a value from 0 through 63. • **fragments**: The policy applies to fragmented packets that contain a non-zero fragment offset. **NOTE:** This option is not applicable to filtering based on source or destination port, TCP flags, and ICMP flags. • **routing**: The policy applies only to IPv6 source-routed packets. **NOTE:** This option is not applicable to filtering based on source or destination port, TCP flags, and ICMP flags.
log	Allows statistics that match the ACL statement to be entered in the syslog.

Unsupported commands and message types

The following commands are not supported for IPv6 ACLs:

- **ipv6-operator flow-label**
- **ipv6-operator fragments** when any protocol is specified. The **fragments** option can be specified only when **permit ipv6** or **deny ipv6** is specified. If you specify **tcp** or any other protocol instead of **ipv6**, the **fragments** keyword cannot be used.
- **ipv6-operator routing** when any protocol is specified. (The same limitation as for **ipv6-operator fragments.**)

Table 14 lists ICMPv6 message types that are not supported for IPv6 ACLs.

TABLE 14 Unsupported ICMPv6 message types

Decimal	ICMP message type
beyond-scope	Destination Unreachable ICMP message, Beyond Scope
destination-unreachable	Destination Unreachable ICMP messages
dscp	Match DSCP value in IPv6 packet
echo-reply	Echo Reply ICMP message
echo-request	Echo Request ICMP message
header	Parameter Problem ICMP Message,Header Error
hop-limit	Time Exceeded ICMP Message, In Transit
log	Log matches against this entry
mld-query	MLD Query Message
mld-reduction	MLD Reduction Message
mld-report	MLD Report Message
nd-na	ND Neighbor Advertisement Message
nd-ns	ND Neighbor Solicitation Message
next-header	Parameter Problem ICMP Message, Next Header
no-admin	Destination Unreachable ICMP Message, Administratively Prohibited
no-route	Destination Unreachable ICMP Message, No Route
packet-too-big	Packet Too Big ICMP Message
parameter-option	Parameter Problem ICMP Message, Option
parameter-problem	Parameter Problem ICMP Messages
port-unreachable	Destination Unreachable ICMP Message, Port Unreachable
reassembly-timeout	Time Exceeded ICMP Message, ReassemblyTimeout
renum-command	Renumber Command ICMP Message
renum-result	Renumber Result ICMP Message
renum-seq-number	Renumber Sequence Number ICMP Message
router-advertisement	Router Advertisement ICMP Message
router-renumbering	All Router Renumbering ICMP Messages
router-solicitation	Router Solicitation ICMP Message
sequence	Sequence number for this entry
time-exceeded	Time Exceeded ICMP Messages
traffic-policy	Attach traffic policy by name
unreachable	Destination Unreachable ICMP message, Address Unreachable

Applying IPv6 ACLs to interfaces

To apply an IPv6 ACL to an interface, enter commands such as the following example.

```
ServerIronADX(config)# interface ethernet 3/1
ServerIronADX(config-if-e100-3/1)# ipv6 traffic-filter access1 in
```

This example applies the IPv6 ACL access1 to incoming IPv6 packets on Ethernet interface 3/1. As a result, Ethernet interface 3/1 denies all incoming packets from the site-local prefix fec0:0:0:2::/64 and the global prefix 2001:db8:100:1::/48 and permits all other incoming packets.

Syntax: ipv6 traffic-filter <*ipv6-acl-name*> **in**

For the <*ipv6-acl-name*> parameter, specify the name of an IPv6 ACL created using the **ipv6 access-list** command.

The **in** keyword applies the specified IPv6 ACL to incoming IPv6 packets on the interface.

Displaying IPv6 ACLs

To display the ACLs configured on a device, enter the **show ipv6 access-list** command.

```
ServerIronADX# show ipv6 access-list
ipv6 access-list v6-acl1: 1 entries
 deny ipv6 any any
ipv6 access-list v6-acl2: 1 entries
 permit ipv6 any any
ipv6 access-list v6-acl3: 2 entries
 deny ipv6 2001:db8:aa:10::/64 any
 permit ipv6 any any
ipv6 access-list v6-acl4: 2 entries
 deny ipv6 2001:db8:aa::/64 any
 permit ipv6 any any
ipv6 access-list v6-acl5: 6 entries
 permit tcp 2001:db8:bb::/64 any
 permit ipv6 2001:db8:bb::/64 any
 permit ipv6 2001:db8:aa:101::/64 any
 permit ipv6 2001:db8:aa:10::/64 2001:db8:aa:102::/64
 permit ipv6 host 2001:db8:aa:10::102 host
2001:db8:aa:101::102
 permit ipv6 any any fragments
```

Syntax: show ipv6 access-list [<*access-list-name*>]

Displaying IPv6 ACLs bound to an interface

To display the ACLs bound to an interface, enter the **show access-list bindings** command.

```
ServerIronADX# show access-list bindings
Access-list binding configuration:
!
interface ethernet 1
ipv6 traffic-filter ipv61 in
!
interface ethernet 2
ipv6 traffic-filter icmp_any in
!
ServerIronADX 1000#
```

Syntax: show access-list bindings

Using an ACL to restrict SSH access

To configure an ACL that restricts SSH access to an IPv6 device, create the named ACL with the ACL statements, and then use the **ssh access-group ipv6** command to restrict SSH access for IPv6.

```
ServerIronADX(config)# ipv6 access-list test2
ServerIronADX(config-ipv6-access-list test2)#  deny ipv6  host 2001:db8:1::1 any
log
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6  2001:db8:1::0/32 any
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6  2001:db8:2::0/32 any
ServerIronADX(config-ipv6-access-list test2)#  permit ipv6 host 2001:db8:3::1 any
ServerIronADX(config-ipv6-access-list test2)# exit
ServerIronADX(config)# ssh access-group ipv6 test2
```

Syntax: [no] ssh access-group ipv6 <*acl-name*>

Using an ACL to restrict Telnet access

To configure an ACL that restricts Telnet access to an IPv6 device, create the named ACL with the ACL statements, and then use the **telnet access-group ipv6** command to restrict Telnet access for IPv6.

```
ServerIronADX(config)# ipv6 access-list test1
ServerIronADX(config-ipv6-access-list test1)#  deny ipv6  host 2001:db8:1::1 any
log
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6  2001:db8:1::0/32 any
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6  2001:db8:2::0/32 any
ServerIronADX(config-ipv6-access-list test1)#  permit ipv6 host 2001:db8:3::1 any
ServerIronADX(config-ipv6-access-list test1)# exit
ServerIronADX(config)# telnet access-group ipv6 test1
```

Syntax: telnet access-group ipv6 <*acl-name*>

Logging IPv6 ACLs

Logging for IPv6 ACLs is disabled by default. To enable logging, enable it for each IPv6 ACL, and then include the logging option in an ACL statement. Logging at both levels must be configured in order for statistics for packets that match the condition to be logged.

```
ServerIronADX(config)# ipv6 access-list acl2
ServerIronADX(config-ipv6-access-list-acl2)# logging-enable
ServerIronADX(config-ipv6-access-list-acl2)# permit tcp host 2001:db8:dabf any eq
http
ServerIronADX(config-ipv6-access-list-acl2)# permit ipv6 any any
```

Syntax: [no] **logging-enable**

NOTE
Syntax for the **log** option in an IPv6 ACL statement are presented in the section "IPv6 ACL syntax" on page 102.

NOTE
Permit logging is not currently supported.

Chapter

7

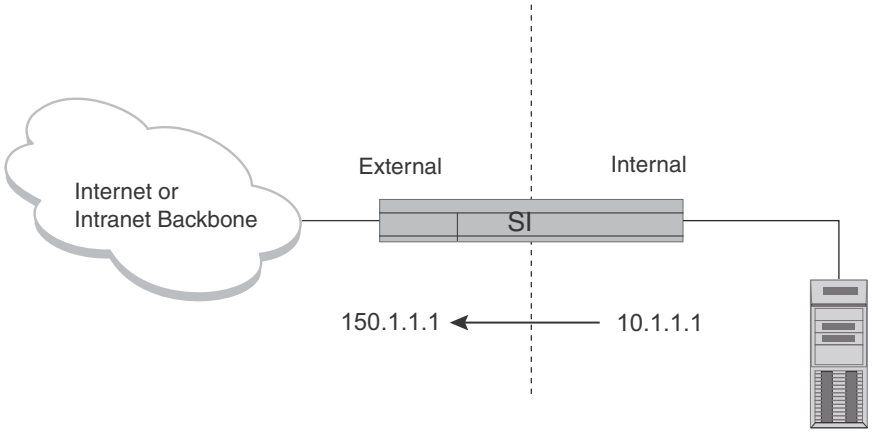
Network Address Translation

In this chapter

- In this chapter 109
- Configuring NAT 110
- Forwarding packets without NAT translation 117
- Forwarding packets without NAT translation 117
- Translation timeouts 118
- Stateless static IP NAT. 120
- IP NAT redundancy. 121
- Displaying NAT information 132
- Clearing NAT entries from the table 137

Network address translation (NAT) allows ServerIron ADX to translate one IP address into another. For example, the ServerIron ADX uses NAT to translate an internal public IP address to be routed over the Internet as shown in Figure 17.

FIGURE 17 Mapping an internal address to an external address



The standard NAT feature described in this section provides translation for hosts attached to private networks behind the ServerIron ADX, and is not to be confused with the address translation features provided for server load balancing (SLB), that is, the standard NAT feature is not related to SLB commands such as **server source-nat**, **server source-nat-ip**, or **server source-ip**.

Configuring NAT

The following types of NAT are supported with ServerIron ADX:

- Static NAT: Maps a specific public IP address (Internet IP address) with a specific private address. Static translation ensures that ServerIron ADX always maps the same public address to a given private address. For example, you can map a specific host (IP address 10.1.1.1) in the private network to always use the same Internet address (150.1.1.1) when communicating outside the private network. The ServerIron ADX supports both inside-to-outside static NAT and outside-to-inside static NAT.
- Dynamic NAT: Maps a group of private addresses with a pool of global IP addresses that you configure. For example, in Figure 18 a global pool within the IP address range of 209.157.1.3/24 through 209.157.1.30/24 is mapped with a private IP subnet 10.10.1.0/24.

  With dynamic NAT, ServerIron ADX uses a round robin technique to select a global IP address to map to a private IP address for every new connection. However, the address selection can get randomized depending on the number of free ports available for the translation.

  In order to reliably de-multiplex return traffic to the internal clients, dynamic NAT uses port address translation (PAT), whereby the ServerIron ADX translates the client's source port into any free port available with the public IP address.

NOTE
You can configure both dynamic and static NAT on the same device. When you configure both types of NAT, static NAT takes precedence over dynamic NAT. Thus, if both static and dynamic NAT entries exist for a private address, the ServerIron ADX will always use the static translation instead of creating a dynamic one.

PAT

Dynamic NAT uses port address translation (PAT). Because there is no one-to-one mapping between private addresses and global addresses, PAT maps a client's IP address and TCP/UDP port to both a global IP address and a TCP/UDP port. In this way, the ServerIron ADX can map many private addresses to the same public address and use TCP/UDP ports to uniquely identify the private hosts.

NOTE
PAT is also called *overloading* an inside global address.

in the following example, using PAT (or overloading) three different private IP addresses with same source ports are mapped to the same global IP address (209.157.1.2), but with different source ports. Thus, the translated source ports help to uniquely identify the reverse session for each dynamic NAT translation.

Inside address	Outside address
10.10.10.2:6000	209.157.1.2:1024
10.10.10.3:6000	209.157.1.2:1025
10.10.10.4:6000	209.157.1.2:1026

Configuring static NAT

Use the **ip nat inside source static** command to explicitly map a private address to a global address. Static NAT ensures a specific host in the private network is always mapped to the global address you specify. For a sample static NAT configuration, see "Static NAT configuration example" on page 116.

For example, to map a private address 10.10.10.69 to a global address 209.157.1.69, you may enter the following command.

```
ServerIronADX(config)# ip nat inside source static 10.10.10.69 209.157.1.69
```

Syntax: [no] ip nat inside source static <*private-ip*> <*global-ip*> [<*priority*>] **[list** <*acl-id*>]

The <*private-ip*> variable specifies the private IP address.

The <*global-ip*> variable specifies the global IP address. The ServerIron ADX supports up to 192 global IP addresses with static NAT.

The <*priority*> variable specifies a value of 1 or 2 and enables static NAT redundancy. A value of 2 means higher priority, and will be the owner of the NAT IP as long as the system is up.

The **list** parameter specifies the access list identified by the <*acl-id*> variable that will permit only the configured TCP or UDP port numbers.

NOTE
You can configure a maximum of 192 static NAT entries on the ServerIron ADX.

NOTE
Static NAT requires unused virtual servers for its operation. For every static NAT entry, two virtual server entries have to be present. The command **show server resource** will show the current number of unused virtual server entries. The command **system-max l4-virtual-server** can be used to increase the virtual server entries if required.

Configuring dynamic NAT

To configure dynamic NAT, perform the following tasks:

- Configure a standard or extended access control list (ACL) for each private address range for which you want to provide NAT. For ACL configuration details, refer to "Configuring rule-based ACLs" on page 65.

  NOTE
  Named ACLS are not supported with NAT. You must use a numbered ACL.

- Configure a pool for each range of consecutive global addresses, which would dynamically map to the private addresses specified in the ACLs. Each pool must contain a range with no gaps. If your global address space has gaps, configure separate pools for each consecutive range within the address space.
- Associate each range of private addresses (identified by a standard or extended ACL) with a pool.

Configuring an address pool

Use the **ip nat pool** command to configure the address pool. For an example, refer to "Dynamic NAT configuration example 1" on page 113.

Syntax: **[no] ip nat pool** <*pool-name*> <*start-ip*> <*end-ip*> **netmask** <*ip-mask*> **| prefix-length** <*length*>

The <*pool-name*> variable specifies the name assigned to the pool. It can be up to 255 characters long and can contain special characters and internal blanks. If you use internal blanks, you must use quotation marks around the entire name.

The <*start-ip*> variable specifies the IP address at the beginning of the pool range. Specify the lowest-numbered IP address in the range.

The <*end-ip*> variable specifies the IP address at the end of the pool range. Specify the highest-numbered IP address in the range.

NOTE
The address range cannot contain any gaps. Make sure you own all the IP addresses in the range. If the range contains gaps, you must create separate pools containing only the addresses you own.

The **netmask** <*ip-mask*> | **prefix-length** <*length*> parameter specifies a classical subnet mask (example: netmask 255.255.255.0) or the length of a CIDR prefix (example: prefix-length 24). The ServerIron ADX supports up to 192 global IP addresses.

NOTE
A ServerIron ADX can have a maximum of 192 global IP addresses, in a single pool or multiple pools.

Associating a range of private addresses with a pool

Use **ip nat inside source list** to associate a private address range with a pool of Internet addresses and enable PAT. For an example, refer to "Dynamic NAT configuration example 1" on page 113.

Syntax: **[no] ip nat inside source list** <*acl-id*> **pool** <*pool-name*> **overload**

The **inside source** keyword specifies that the translation applies to private addresses sending traffic to the Internet (inside source).

The **list** <*acl-id*> parameter specifies a standard or extended ACL. Named ACLS are not supported with NAT. You must use a numbered ACL.

The **pool** <*pool-name*> parameter specifies the pool name. You must create the pool before you can use it with this command.

The **overload** keyword enables PAT with the dynamic NAT and is required in order to configure dynamic NAT.

Enabling IP NAT

When a ServerIron ADX is configured with switch code, NAT is enabled globally but when it is configured with router code, it is enabled per-interface.

NOTE
ServerIron ADX does not support IP NAT inside and outside on the same physical interface.

Enabling IP NAT globally

The following command enables IP NAT globally.

```
ServerIronADX(config)# ip nat inside
```

Syntax: [no] ip nat inside

Enabling IP NAT per-interface

When enabled per-interface, IP NAT must be enabled exclusively “inside” or “outside” on a physical or virtual interface as shown in the following example.

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-e1000-1/5) ip nat inside
```

Syntax: [no] ip nat [inside | outside]

The **inside** parameter configures the interface as an IP NAT inside interface.

The **outside** parameter configures the interface as an IP NAT outside interface.

NAT configuration examples

The following sections provide both dynamic and static NAT configuration examples.

Dynamic NAT configuration example 1

This example describes the dynamic NAT configuration for a ServerIron ADX running switch code as shown in Figure 18.

FIGURE 18 Dynamic NAT translating inside host addresses to a pool of global addresses

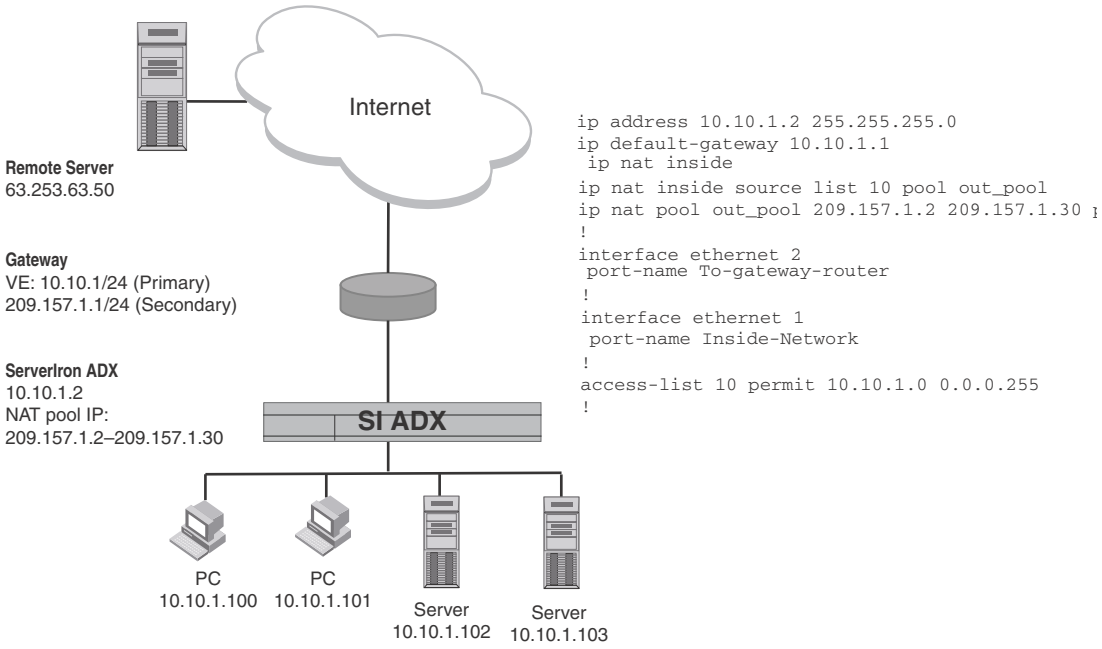


Figure 18 shows a dynamic NAT configuration on a ServerIron ADX, running with switch code. The ServerIron ADX is connected to the Internet through a router. The private network—also referred to as the *inside network*—consists of IP addresses in the range 10.10.1.2 through 10.10.1.254, with a 24-bit subnet mask. A pool of global addresses in the range of 209.157.1.2 through 209.157.1.30 is configured on the ServerIron ADX, which is used to translate the private network.

Minimum required commands for dynamic NAT configuration with switch code:

1. Configure a numbered ACL and permit the IP addresses on the inside.

   ```
   ServerIronADX(config)# access-list 101 permit ip 10.10.1.0/24 any
   ```

   Make sure you specify the **permit** parameter in the ACL, rather than the **deny** parameter. If you specify the **deny** parameter, the ServerIron ADX will not provide NAT for the addresses.

2. Define the global address pool.

   ```
   ServerIronADX(config)# ip nat pool global_pool 209.157.1.2 209.157.1.30
   prefix-length 24
   ```

3. Tie the inside source list to the global pool and enable PAT (overload).

   ```
   ServerIronADX(config)# ip nat inside source list 101 pool global_pool overload
   ```

4. Enable dynamic NAT globally on the ServerIron ADX.

   ```
   ServerIronADX(config)# ip nat inside
   ```

5. Assign a secondary IP address from the public IP subnet on the next hop router's interface, which is connected to the ServerIron ADX, as shown in Figure 18.

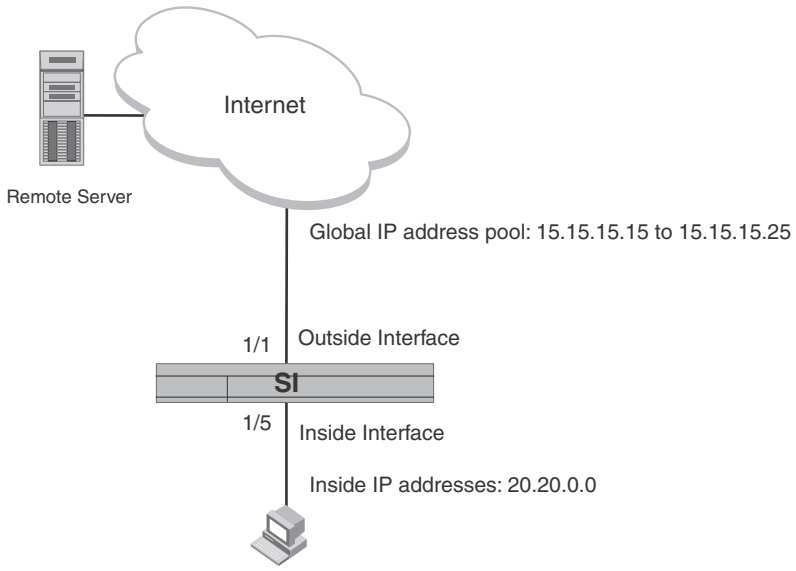
You may log into the barrel processor (BP) using rconsole and verify that the translation is working correctly. Enter a command such as the one in the following example.

```
ServerIronADX# rconsole 1/1
ServerIronADX1/1#show ip nat statistic
ServerIronADX1/1#show ip nat translation
```

Dynamic NAT configuration example 2

Figure 19 shows dynamic NAT configuration on a ServerIron ADX running on router code, translating inside hosts in the 20.20.0.0/16 network to a pool of global addresses in the 15.15.15.15/24 network.

FIGURE 19 Dynamic NAT translating inside host addresses to a pool of global addresses



In the example shown in Figure 19, the ServerIron ADX is acting as a gateway between the private network and the Internet. The private network—also referred to as the *inside network*—consists of IP addresses in the range 20.20.0.0 through 20.20.255.254, with a 16-bit subnet mask. A pool of global IP addresses in the range of 15.15.15.15 through 15.15.15.25 is configured on the ServerIron ADX, which is used to translate the private network.

1. Identify internal and external interfaces on the ServerIron ADX and assign IP addresses to them. In this example, interfaces 1/5 and 1/1 are configured as *inside* and *outside* interfaces respectively as shown:

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-e1000-1/5) ip address 20.20.50.1 255.255.0.0
ServerIronADX(config-if-e1000-1/5) ip nat inside
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-e1000-1/1) ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-e1000-1/1) ip nat outside
```

2. Configure an access list to permit traffic from inside hosts in the 20.20.0.0 network as shown.

```
ServerIronADX(config)# access-list 1 permit 20.20.0.0 0.0.255.255
```

3. Create a pool of global IP addresses from 15.15.15.15 to 15.15.15.25, named *p1*.

```
ServerIronADX(config)# ip nat pool p1 15.15.15.15 15.15.15.25 prefix-len 24
```

4. Bind the inside source access list *1* to the public address pool *p1* and enable PAT (overload).

```
ServerIronADX(config)# ip nat inside source list 1 pool p1 overload
```

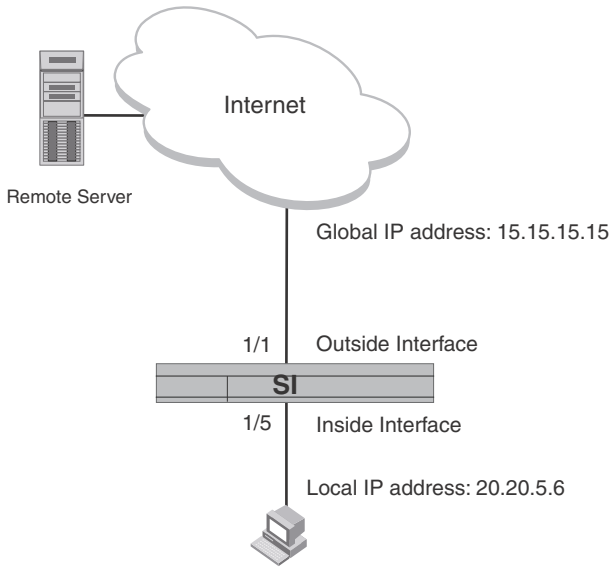
5. You may log into the barrel processor (BP) using the rconsole and verify that the translation is working correctly. Enter commands like those in the following example.

```
ServerIronADX# rconsole 1/1
ServerIronADX1/1# show ip nat statistic
ServerIronADX1/1# show ip nat translation
```

Static NAT configuration example

The following example describes how to configure a static NAT entry for Inside-to-outside and outside-to-inside translation for the network shown in Figure 20.

FIGURE 20 Example of a static NAT configuration with ServerIron ADX



Configured for inside-to-outside translation

In this example, the ServerIron ADX is configured to translate the local host IP address 20.20.5.6 to the unique global address 15.15.15.15.

Assign IP addresses to the interfaces and, on router code, enable NAT on both the inside and the outside interfaces. The interfaces can also be virtual interfaces (not necessarily physical interfaces).

```
ServerIronADX(config)# interface ethernet 1/5
ServerIronADX(config-if-1/5)# ip address 20.20.50.1 255.255.0.0
ServerIronADX(config-if-1/5)# ip nat inside
ServerIronADX(config-if-1/1)# int eth 1/1
ServerIronADX(config-if-1/1)# ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-1/1)# ip nat outside
```

On switch code, enable NAT globally.

```
ServerIronADX(config)# ip nat inside
```

The following command configures the ServerIron ADX to translate IP packets with a local IP address of 20.20.5.6 to the global IP address 15.15.15.15.

```
ServerIronADX(config)# ip nat inside source static 20.20.5.6 15.15.15.15
```

Configured for outside-to-inside translation

To configure the ServerIron ADX shown in Figure 20 for outside-to-inside translation as well, the only requirement is that outside interface must be configured with an additional IP address in the public subnet, in this case 15.15.15.0/24 as shown in the following example.

```
ServerIronADX(config)# interface ethernet 1/1
ServerIronADX(config-if-e1000-1/1) ip address 30.30.0.1 255.255.0.0
ServerIronADX(config-if-e1000-1/1) ip address 15.15.15.100 255.255.0.0
ServerIronADX(config-if-e1000-1/1) ip nat outside
```

Forwarding packets without NAT translation

By default, if the ServerIron ADX receives a non-SYN packet for a TCP flow from an internal NAT client and no existing NAT session is found, the ServerIron ADX will drop that packet.

You can optionally configure the ServerIron ADX to forward such packets without NAT translation by entering the following command.

```
ServerIronADX(config)# nat-forward-no-session
```

Syntax: [no] nat-forward-no-session

The **nat-forward-no-session** command is required in some special cases (such as when dynamic NAT is used with VIP overlap) when the ServerIron ADX would drop packets because it cannot locate the corresponding NAT sessions. This occurs in cases where the remote client open connections to a real server directly and the ServerIron ADX routes the initial SYN packet to the real server. When the SYN-ACK packet subsequently arrives from the real server, the ServerIron ADX checks the session table to locate the corresponding session. However, because the SYN-ACK packet does not belong to the an session from the real server itself, there will be no corresponding session. Therefore, the ServerIron ADX will not find the session table entry and would drop it. If the **nat-forward-no-session** command is configured, the SYN-ACK packet is not dropped and is forwarded to remote client.

IP NAT with VIP overlap

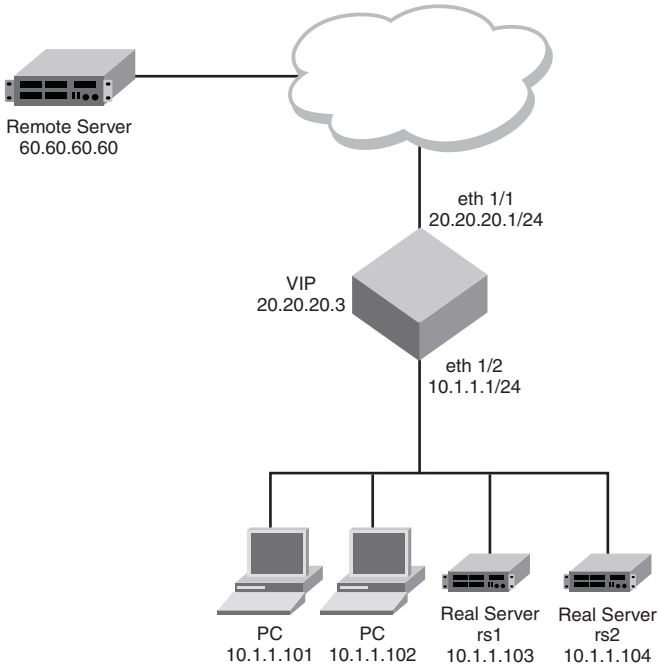
The ServerIron ADX can be configured to use an IP address assigned to a virtual server as a dynamic NAT pool IP.

In this configuration, the connections initiated from inside going out will be translated and will take the source IP of the virtual server. The return traffic to these already established connections will be routed back to the host that initiated this connection. At the same time, a connection initiated by an outside host coming in to the virtual server, will be load balanced and sent to a real server.

The access list (ACL) defining the traffic for this scenario can contain both real servers already bound to the virtual server, as well as any other hosts on the same subnet. Brocade recommends that for this scenario, the dynamic NAT pool should only contain a single IP address, that is, the virtual server IP address.

Figure 21 illustrates an example where IP NAT is configured with VIP overlap.

FIGURE 21 Example of IP NAT with VIP overlap



In this example, any host on the inside network (10.1.1.0), which has to initiate a connection to the remote host, will get translated to the virtual server IP address (20.20.20.3). On the other hand, any connection coming to the virtual server from the outside network will get load balanced to one of the real servers (rs1 and rs2).

NOTE
Static NAT is not supported with VIP overlap.

Translation timeouts

The NAT translation table contains all active NAT translation entries on the device. An active entry is one which the ServerIron ADX creates for a private address when the client at that address sends traffic to the outside network.

The ServerIron ADX performs the following steps to provide NAT translation for a source IP address:

- If an active NAT entry exists for the session in the translation table, the ServerIron ADX uses that entry.
- If no active NAT entry exists for the session in the translation table, the ServerIron ADX creates a new entry and places it into the translation table.

Each NAT entry remains in the translation table until the entry ages out.

Configuring the NAT translation aging timer

Use the **ip nat translation** command to alter the NAT translation aging timer. For example, the following command increases the NAT translation timeout value for TCP sessions to 30 minutes:

```
ServerIronADX(config)# ip nat translation tcp-timeout 1800
```

Syntax: [no] ip nat translation dns-timeout | finrst-timeout | icmp-timeout | syn-timeout | tcp-timeout | udp-timeout <*secs*>| maximum

The **dns-timeout** keyword indicates connections to a DNS server. The default is 120 seconds.

The **finrst-timeout** keyword identifies TCP FIN (finish) and RST (reset) packets, which normally terminate TCP connections. The default is 120 seconds. This timer is not related to **tcp-timeout**, which applies to packets to or from a host address that is mapped to an global IP address and a TCP port number (PAT feature). The **finrst-timeout** applies to packets that terminate a TCP session, regardless of the host address or whether PAT is used.

The **icmp-timeout** keyword indicates timeout for ICMP NAT flows.

The **syn-timeout** keyword indicates timeout for TCP NAT flows after a SYN.

The **tcp-timeout** keyword indicates dynamic NAT entries that use PAT based on TCP port numbers. The default is 120 seconds. This timer applies only to TCP sessions that do not end “gracefully”, with a TCP FIN or TCP RST.

The **udp-timeout** keyword indicates dynamic NAT entries that use PAT based on UDP port numbers. The default is 120 seconds.

The <*secs*> parameter specifies number of seconds, from 0 through 3600. Use **maximum** to set the maximum timeout value. For example, 3,600 seconds.

After the configuration of a NAT timeout the ServerIron ADX handles NAT sessions as follows:

- Existing session entries of the modified protocol remain in the translation table.
- If an existing session entry sends or receives traffic, the NAT timeout value for that entry is updated to the new configured value.
- If an existing session entry does not process any traffic, it continues to age out in accordance with the old NAT timeout value.
- When the NAT timeout (age of the session) expires, forward and reverse sessions on the ServerIron ADX are deleted with no further actions. If traffic is received on this flow, the ServerIron ADX drops the packets because it will not find any sessions related to that particular flow.

Disabling IP NAT sticky behavior

By default, when a dynamic IP NAT client initiates traffic, the ServerIron ADX selects a NAT pool IP and creates a sticky session, which associates this client's IP with the same NAT pool IP. For all subsequent flows from the client, the same NAT pool IP is selected as long as the sticky session exists. However, under certain heavy traffic conditions, the NAT pool IP might run out of ports, resulting in dropped connections.

To override this behavior and allow the ServerIron ADX to select a different NAT pool IP each time for the same client, enter the following command:

```
ServerIronADX(config)# ip nat disable-sticky
```

Syntax: [no] ip nat disable-sticky

Deleting IP NAT sticky sessions

By default, when a dynamic IP NAT client initiates traffic, the ServerIron ADX selects a NAT pool IP and creates a sticky session, which associates this client's IP with the same NAT pool IP. For all subsequent flows from the client, the same NAT pool IP is selected as long as the sticky session exists.

However, under certain heavy traffic conditions, the NAT pool IP might run out of ports, resulting in dropped connections. Enter the **server nat-sticky-delete** command to override this behavior:

```
ServerIronADX(config)# server nat-sticky-delete
```

Syntax: [no] server nat-sticky-delete

If a new client connection request arrives and the NAT pool IP address in a sticky session has run out of available ports, the ServerIron ADX (if configured with the **server nat-sticky-delete** command) will delete the existing sticky session, select a new NAT pool IP, and create a new sticky session for the source IP-NAT pool IP pair. Any existing session with this client will continue to exist, however, all new connections will use the new sticky session.

If the **server nat-sticky-delete** command is not configured, the ServerIron ADX will generate a port unreachable ICMP message.

Stateless static IP NAT

The ServerIron ADX creates sessions for static NAT traffic by default. You can disable this behavior using the following command.

```
ServerIronADX(config)# ip nat stateless
```

Syntax: [no] ip nat stateless

For the **ip nat stateless** command to work, the **ip nat inside source static** command must already be configured.

```
ServerIronADX(config)# ip nat inside source static 10.45.16.103 157.29.56.3
ServerIronADX(config)# ip nat stateless
```

NOTE
IP NAT stateless only supports TCP, UDP, and ICMP traffic. FTP, RTSP, and other similar complex protocols are not currently supported.

NOTE
You must reload the ServerIron ADX whenever changes are made to a running IP NAT configuration.

IP NAT redundancy

The ServerIron ADX supports static and dynamic IP NAT in redundant/HA environments using Hot-Standby mode with switch code, or Sym-Active (Active-Active) mode using VRRP-E with router code. Further information on VRRP-E can be found in the *ServerIron ADX Switch and Router Guide* under section “Configuring VRRP and VRRP-E".

In order to configure IP NAT redundancy with a Hot-Standby deployment, you must specify the active ServerIron ADX as the owner of the global NAT IP address or pool. Similarly, in Sym-Active mode, the ServerIron ADX with the higher VRRP-E backup priority must be specified as the NAT IP address owner. The owner of the NAT IP address is responsible for responding to ARP requests directed to the redundant public IP address.

Configuring static NAT entry in a HA setup

In a high availability (HA) deployment, the static NAT entry is supported by both of the ServerIron ADX HA partners.

To configure a static NAT entry, use the **ip nat inside source static** command with the <*priority-value*> parameter as shown in the following example:

```
ServerIronADX(config)# ip nat inside source static 10.10.10.10 63.32.23.1 2
```

Syntax: [no] ip nat inside source static <*ip-addr1*> <*ip-addr2*> <*priority-value*>

The <*priority-value*> is used to determine the owner of the static NAT IP address. A value of either *1* or *2* can be configured with 2 being the higher priority. The ServerIron ADX configured with the higher priority, is the owner of the NAT IP address as long as it stays up.

Configuring dynamic NAT pool in a HA setup

To configure a dynamic NAT pool which will be supported on a ServerIron ADX pair in HA setup, enter the **ip nat pool** command with **port-pool-range** operand as shown in the following example:

```
ServerIronADX(config)# ip nat pool pool1 63.23.1.2 63.23.1.4 prefix 24
ServerIronADX(config)# ip nat pool pool1 port-pool-range 2
```

Syntax: [no] ip nat pool <pool-name> port-pool-range <*pool range value*>

The **port-pool-range** <*pool range value*> parameter is similar to the priority value for static NAT, except that it also determines the range of source ports allocated for the NAT IP, which prevents source port collision.

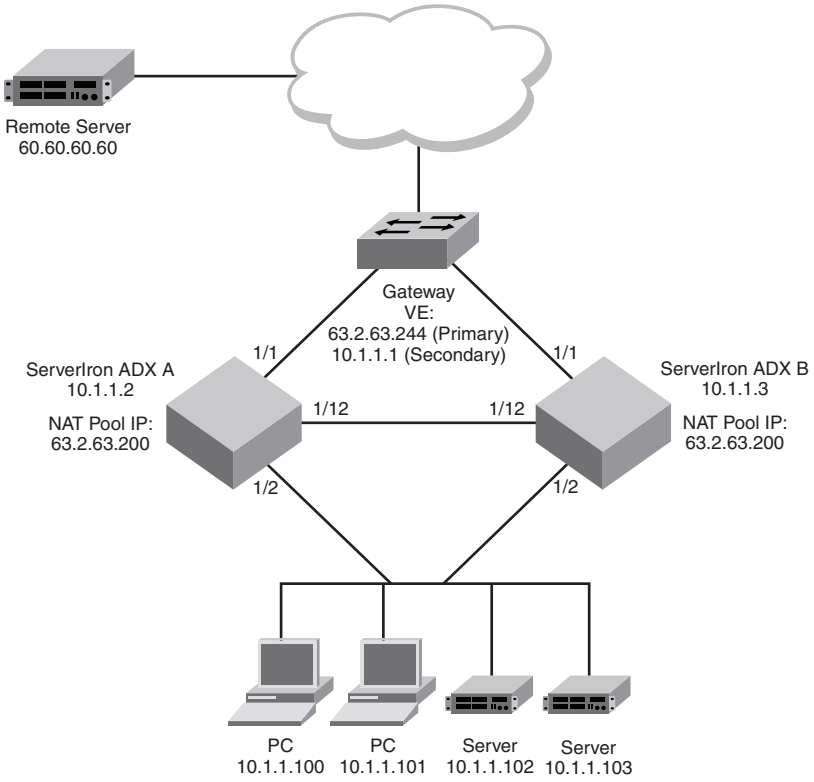
The <*pool range value*> can be 1 or 2; where 2 has the higher priority. The ServerIron ADX configured with the higher priority 2 will be the owner of the NAT pool addresses. It also means the source ports allocated for the NAT IPs are from the higher range.

NOTE
A distribution of port ranges is not required for static NAT, as it does not involve PAT.

Configuring dynamic NAT redundancy in Hot-Standby mode

Follow these steps to enable the minimum required configuration for dynamic NAT in Hot-Standby mode, as shown in Figure 22.

FIGURE 22 Minimum required configuration for dynamic NAT in Hot-Standby setup



1. On both ServerIron ADX units, place the untagged Hot-Standby port (sync-link) in a separate VLAN and disable STP:

```
ServerIronADX(config)# vlan 99 by port
ServerIronADX(config-vlan-99)# untagged ethernet 1/12
ServerIronADX(config-vlan-99)# no spanning-tree
```

   Placing the Hot Standby port in a separate VLAN prevents unnecessary traffic from going over the sync-link. With Hot Standby, you cannot have spanning-tree configured anywhere on any link connected between the two ServerIron ADX units.

2. Configure the server backup port, shared chassis-MAC address between the ServerIron ADX units, and any connected router-ports:

```
ServerIronADX(config)# server backup ethernet 1/12 00e0.5201.0c72 vlan-id 99
ServerIronADX(config)# server router-ports ethernet 1/1
```

   The **server backup ethernet** command must be configured exactly the same on both devices. The <*mac-addr*> variable specifies the chassis-MAC address of one of the ServerIron ADX units. Be sure to use a chassis MAC address from one of the two devices, not the MAC address of one of the backup ports. Use show chassis to locate the chassis MAC address.

   The **server router-ports** command enables the ServerIron ADX to count the number of upstream (or downstream) router ports connected to the device.

3. Configure a standard or extended ACL identifying each private address range for which you wish to provide NAT:

```
ServerIronADX(config)# access-list 10 permit 10.1.1.0 0.0.0.255
```

4. Configure a dynamic NAT pool on each ServerIron ADX, and assign device ownership to the NAT pool. In this example, ServerIron ADX A is assigned as the NAT Pool owner, and therefore takes the higher priority value:

```
SI_A(config)# ip nat pool P1 63.2.63.200 63.2.63.200 prefix-len 24
SI_A(config)# ip nat pool P1 port-pool-range 2

SI_B(config)# ip nat pool P1 63.2.63.200 63.2.63.200 prefix-len 24
SI_B(config)# ip nat pool P1 port-pool-range 1
```

5. Tie the inside source ACL to the dynamic NAT pool and enable PAT (overload)

```
ServerIronADX(config)# ip nat inside source list 10 pool P1 overload
ServerIronADX(config)# ip nat inside
```

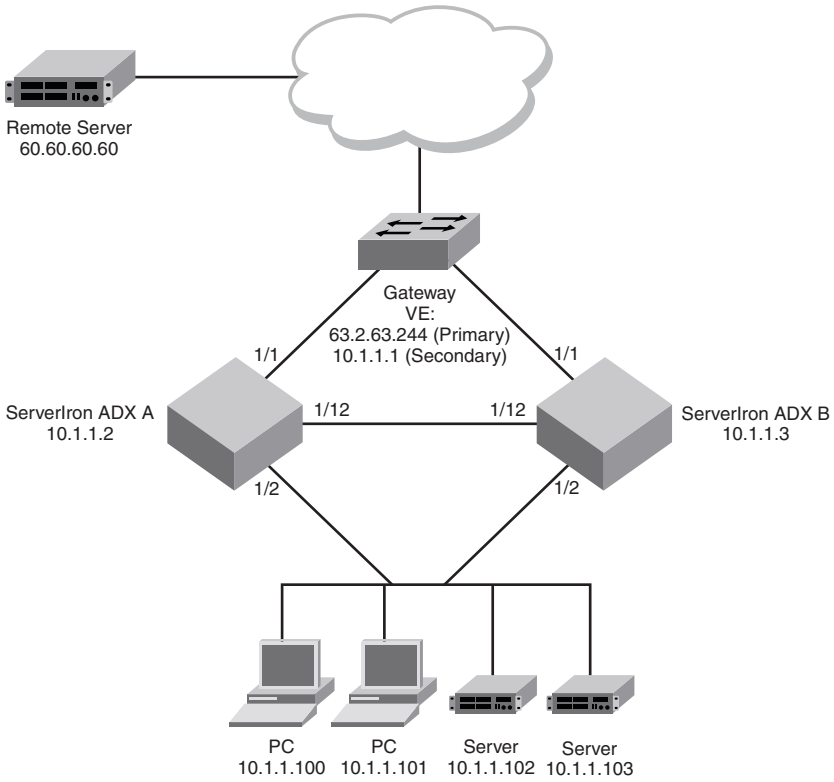
6. Configure the inside interface of the default gateway router with an IP address from both the dynamic NAT pool subnet, as well as the inside network. The reason is that the default gateway should be ping-able from both the ServerIron ADX switch as well as from the private client subnet. For example:

```
Router(config)# interface ve 2
Router(config-VE)# ip address 63.2.63.44 255.255.255.0
Router(config-VE)# ip address 10.1.1.1 255.255.255.0
```

Configuring static NAT redundancy in Hot-Standby mode

Follow these steps to enable the minimum required configuration for static NAT in Hot-Standby mode, as shown inFigure 23.

FIGURE 23 Minimum required configuration for static NAT in Hot-Standby setup



1. On both ServerIron ADX units, place the untagged Hot-Standby port (sync-link) in a separate VLAN and disable STP:

```
ServerIronADX(config)# vlan 99 by port
ServerIronADX(config-vlan-99)# untagged ethernet 1/12
ServerIronADX(config-vlan-99)# no spanning-tree
```

   Placing the Hot Standby port in a separate VLAN prevents unnecessary traffic from going over the sync-link. With Hot Standby, you cannot have spanning-tree configured anywhere on any link connected between the two ServerIron ADX units.

2. Configure the server backup port, shared chassis-MAC address between the ServerIron ADX units, and any connected router-ports:

```
ServerIronADX(config)# server backup ethernet 1/12 00e0.5201.0c72 vlan-id 99
ServerIronADX(config)# server router-ports ethernet 1/1
```

   The **server backup ethernet** command must be configured exactly the same on both devices. The <mac-addr> variable specifies the chassis-MAC address of one of the ServerIron ADX units. Be sure to use a chassis MAC address from one of the two devices, not the MAC address of one of the backup ports. Use show chassis to locate the chassis MAC address.

   The **server router-ports** command enables the ServerIron ADX to count the number of upstream (or downstream) router ports connected to the device.

3. Configure the static NAT entries on each ServerIron ADX, and assign device ownerships to the NAT entries. In this example, ServerIron ADX A is assigned as the NAT owner, and therefore takes the higher priority value:

```
SI_A(config)# ip nat inside source static 10.1.1.100 63.2.63.100 2
SI_A(config)# ip nat inside source static 10.1.1.101 63.2.63.101 2
SI_A(config)# ip nat inside source static 10.1.1.102 63.2.63.102 2
SI_A(config)# ip nat inside source static 10.1.1.103 63.2.63.103 2

SI_B(config)# ip nat inside source static 10.1.1.100 63.2.63.100 1
SI_B(config)# ip nat inside source static 10.1.1.101 63.2.63.101 1
SI_B(config)# ip nat inside source static 10.1.1.102 63.2.63.102 1
SI_B(config)# ip nat inside source static 10.1.1.103 63.2.63.103 1
```

4. Enable static NAT on both devices:

```
ServerIronADX(config)# ip nat inside
```

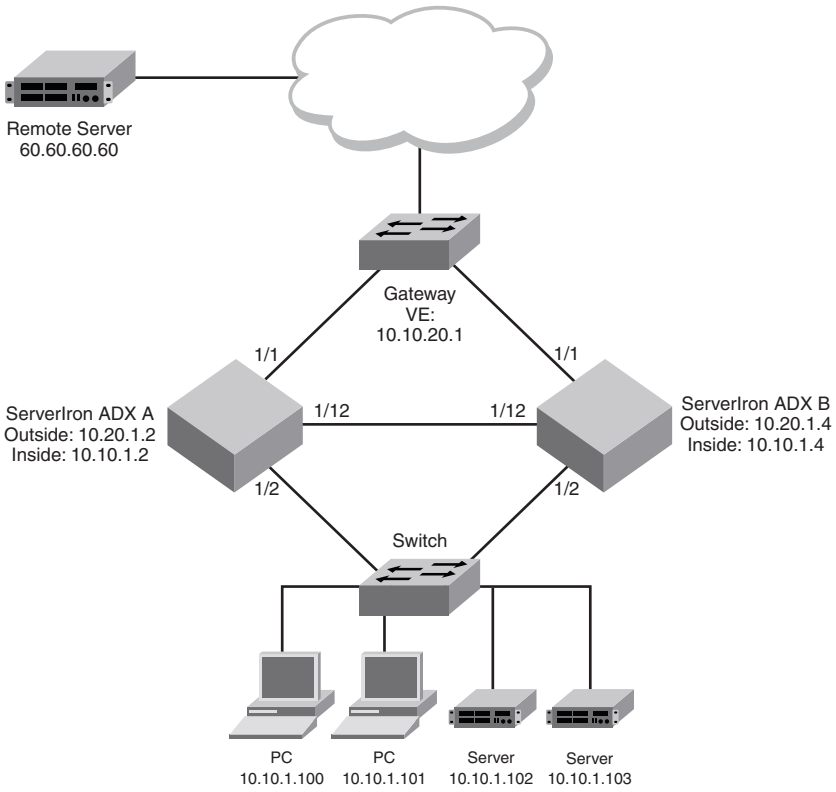
5. Configure the inside interface of the default gateway router with an IP address from both the static NAT subnet, as well as the inside network. The reason is that the default gateway should be ping-able from both the ServerIron ADX switch as well as from the private client subnet. For example:

```
Router(config)# interface ve 2
Router(config-VE)# ip address 63.2.63.44 255.255.255.0
Router(config-VE)# ip address 10.1.1.1 255.255.255.0
```

Configuring dynamic NAT redundancy in Sym-Active (Active-Active) mode

The Sym-Active (Active-Active) mode is available for IP NAT in router code, and it is configured in combination with VRRP-E. Also, IP NAT is VIP group-aware and requires NAT port pools to be configured in the VIP group, which is then tied to the VRRP-E configuration. Follow these steps to enable the minimum required configuration for dynamic NAT in Sym-Active mode, as shown in Figure 24.

FIGURE 24 Minimum required configuration for dynamic NAT in Sym-Active setup



1. On both ServerIron ADX units, place the untagged sync-link in a separate VLAN and disable STP:

```
ServerIronADX(config)# vlan 99 by port
ServerIronADX(config-vlan-99)# untagged ethernet 1/12
ServerIronADX(config-vlan-99)# static-mac-address 00e0.52ee.6900 ethernet 1/12
ServerIronADX(config-vlan-99)# no spanning-tree
```

   Placing the sync-link in a separate VLAN prevents unnecessary traffic from going over the sync-link. With HA, you cannot have spanning-tree configured anywhere on any link connected between the two ServerIron ADX units. The **static-mac-address** command is configured with the MAC address of the interface at the other end of the sync-link.

2. Configure the **server active-active-port** command on both ServerIron ADX units, along with and any connected router ports. Also, enable VRRP-E on the ServerIron ADX:

```
ServerIronADX(config)# server active-active-port ethernet 1/12 vlan-id 99
ServerIronADX(config)# server router-ports ethernet 1/1
ServerIronADX(config)# router vrrp-extended
```

   The **server router-ports** command enables the ServerIron ADX to count the number of upstream (or downstream) router ports connected to the device.

3. Identify the inside and outside networks and assign them to different VLANs. VRRP-E requires the interface on which you configure a virtual router ID (VRID) to have an IP interface that is in the same subnet as the VRID address.

```
ServerIronADX(config)# vlan 100
ServerIronADX(config-vlan-100)# untagged ethernet 1/1
ServerIronADX(config-vlan-100)# router-interface ve 1
ServerIronADX(config-vlan-100)# exit
ServerIronADX(config)# vlan 200
ServerIronADX(config-vlan-200)# untagged ethernet 1/2
ServerIronADX(config-vlan-200)# router-interface ve 2
ServerIronADX(config-vlan-200)# exit
ServerIronADX-A(config)# interface ve 1
ServerIronADX-A(config-ve-1)# 10.10.20.2 255.255.255.0
ServerIronADX-A(config-ve-1)# ip nat outside
ServerIronADX-A(config-ve-1)# exit
ServerIronADX-A(config)# interface ve 2
ServerIronADX-A(config-ve-2)# 10.10.10.2 255.255.255.0
ServerIronADX-A(config-ve-2)# ip nat inside
ServerIronADX-A(config-ve-2)# exit

ServerIronADX-B(config)# interface ve 1
ServerIronADX-B(config-ve-1)# 10.10.20.4 255.255.255.0
ServerIronADX-A(config-ve-1)# ip nat outside
ServerIronADX-B(config-ve-1)# exit
ServerIronADX-B(config)# interface ve 2
ServerIronADX-B(config-ve-2)# 10.10.10.4 255.255.255.0
ServerIronADX-A(config-ve-2)# ip nat inside
ServerIronADX-B(config-ve-2)# exit
```

4. Configure a standard or extended ACL identifying each private address range for which you wish to provide NAT:

```
ServerIronADX(config)# access-list 10 permit 10.10.1.0 0.0.0.255
```

5. Configure a dynamic NAT pool on each ServerIron ADX, and assign device ownership to the NAT pool. In this example, ServerIron ADX A is assigned as the NAT pool owner, and therefore takes the higher priority value:

```
ServerIronADX-A(config)# ip nat pool P1 10.10.20.21 10.10.20.40 prefix-len 24
ServerIronADX-A(config)# ip nat pool P1 port-pool-range 2

ServerIronADX-B(config)# ip nat pool P1 10.10.20.21 10.10.20.40 prefix-len 24
ServerIronADX-B(config)# ip nat pool P1 port-pool-range 1
```

6. Tie the inside source ACL to the dynamic NAT pool and enable PAT (overload)

```
ServerIronADX(config)# ip nat inside source list 10 pool P1 overload
```

7. Configure the VRRP-E parameters. The IP address configured with the **ip-address** command is the address that will be backed up by VRRP-E. The **track-port** commands enable tracking on the interfaces on the other side of the ServerIron ADX that complete the link for the VRID. For example, traffic that is addressed to VRID 1 enters the ServerIron ADX through VE interface 1 and leaves the ServerIron ADX through VE interface 2. Under normal circumstances, if VE interface 2 goes down, VRID 1 remains active.

   When you track interfaces for a VRID, if the state of one of the tracked interfaces changes, the ServerIron ADX associates the change with the VRID interface. For example, if virtual routing interface 2 goes down, the ServerIron ADX associates this state change with VRID 1 and causes VRRP-E to fail over the VRID to the other ServerIron ADX. The ServerIron ADX on which you configure the higher VRRP-E backup priority becomes the default master for the VRID, while other ServerIron ADX becomes the backup.

   In this example, ServerIron ADX A is configured as the default master for the HA setup.

```
ServerIronADX-A(config)# interface ve 1
ServerIronADX-A(config-ve-1)# ip vrrp-extended vrid 1
ServerIronADX-A(config-ve-1-vrid-1)# backup priority 200 track-priority 10
ServerIronADX-A(config-ve-1-vrid-1)# ip-address 10.10.20.3
ServerIronADX-A(config-ve-1-vrid-1)# enable
ServerIronADX-A(config-ve-1-vrid-1)# track-port ethernet 1/2
ServerIronADX-A(config-ve-1-vrid-1)# track-port ve 2
ServerIronADX-A(config-ve-1-vrid-1)# exit
ServerIronADX-A(config-ve-1)# exit
ServerIronADX-A(config)# interface ve 2
ServerIronADX-A(config-ve-2)# ip vrrp-extended vrid 2
ServerIronADX-A(config-ve-2-vrid-2)# backup priority 200 track-priority 10
ServerIronADX-A(config-ve-2-vrid-2)# ip-address 10.10.10.3
ServerIronADX-A(config-ve-2-vrid-2)# enable
ServerIronADX-A(config-ve-2-vrid-2)# track-port ethernet 1/1
ServerIronADX-A(config-ve-2-vrid-2)# track-port ve 1
ServerIronADX-A(config-ve-2-vrid-2)# exit
ServerIronADX-A(config-ve-2)# exit

ServerIronADX-B(config)# interface ve 1
ServerIronADX-B(config-ve-1)# ip vrrp-extended vrid 1
ServerIronADX-B(config-ve-1-vrid-1)# backup priority 100 track-priority 10
ServerIronADX-B(config-ve-1-vrid-1)# ip-address 10.10.20.3
ServerIronADX-B(config-ve-1-vrid-1)# enable
ServerIronADX-B(config-ve-1-vrid-1)# track-port ethernet 1/2
ServerIronADX-B(config-ve-1-vrid-1)# track-port ve 2
ServerIronADX-B(config-ve-1-vrid-1)# exit
ServerIronADX-B(config-ve-1)# exit
ServerIronADX-B(config)# interface ve 2
ServerIronADX-B(config-ve-2)# ip vrrp-extended vrid 2
ServerIronADX-B(config-ve-2-vrid-2)# backup priority 100 track-priority 10
ServerIronADX-B(config-ve-2-vrid-2)# ip-address 10.10.10.3
ServerIronADX-B(config-ve-2-vrid-2)# enable
ServerIronADX-B(config-ve-2-vrid-2)# track-port ethernet 1/1
ServerIronADX-B(config-ve-2-vrid-2)# track-port ve 1
ServerIronADX-B(config-ve-2-vrid-2)# exit
ServerIronADX-B(config-ve-2)# exit
```

8. Configure a VIP group on each device and use the **ip-nat-pool** command to add the dynamic NAT pool under the VIP group.

```
ServerIronADX(config)# server vip-group 1
ServerIronADX(config-vip-group-[1])# ip-nat-pool P1
```

9. Once the dynamic NAT pools are defined as VIP group members, the VIP group must be added to a VRID. It must be added to the outside interface (the interface having ip nat outside configured):

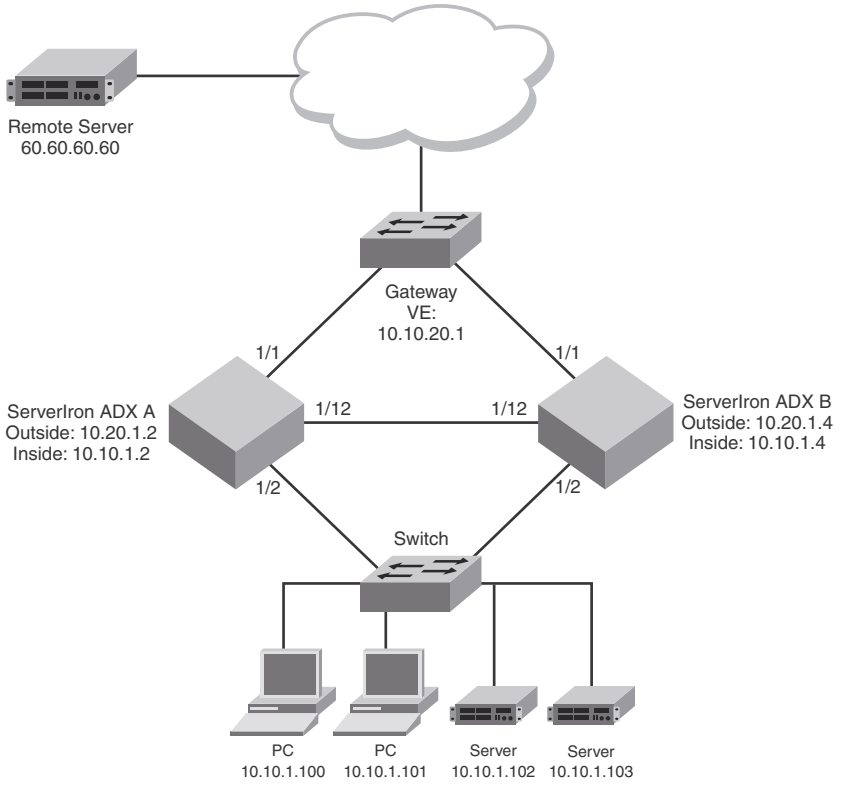
```
ServerIronADX(config)# interface ve1
ServerIronADX(config-ve-1)# ip vrrp vrid 1
ServerIronADX(config-ve-1-vrid-1)# vip-group 1
```

Note that each VIP group can have only one VRID associated with it. Also, each virtual IP address can belong to only one VIP group.

Configuring Static NAT redundancy in Sym-Active (Active-Active) mode:

The Sym-Active (Active-Active) mode is available for IP NAT in router code, and it is configured in combination with VRRP-E. Also, IP NAT is VIP group-aware and requires NAT port pools to be configured in the VIP group, which is then tied to the VRRP-E configuration. Follow these steps to enable the minimum required configuration for static NAT in Sym-Active mode, as shown in Figure 25.

FIGURE 25 Minimum required configuration for static NAT in Sym-Active setup



1. On both ServerIron ADX units, place the untagged sync-link in a separate VLAN and disable STP:

```
ServerIronADX(config)# vlan 99 by port
ServerIronADX(config-vlan-99)# untagged ethernet 1/12
ServerIronADX(config-vlan-99)# static-mac-address 00e0.52ee.6900 ethernet 1/12
ServerIronADX(config-vlan-99)# no spanning-tree
```

   Placing the sync-link in a separate VLAN prevents unnecessary traffic from going over the sync-link. With HA, you cannot have spanning-tree configured anywhere on any link connected between the two ServerIron ADX units. The **static-mac-address** command is configured with the MAC address of the interface at the other end of the sync-link.

2. Configure the server **active-active-port** command on both ServerIron ADX units, along with and any connected router-ports. Also enable VRRP-E on the ServerIron ADX:

```
ServerIronADX(config)# server active-active-port ethernet 1/12 vlan-id 99
ServerIronADX(config)# server router-ports ethernet 1/1
ServerIronADX(config)# router vrrp-extended
```

The **server router-ports** command enables the ServerIron ADX to count the number of upstream (or downstream) router ports connected to the device.

3. Identify the inside and outside networks and assign them to different VLANs. VRRP-E requires the interface on which you configure a virtual router ID (VRID) to have an IP interface that is in the same subnet as the VRID address.

```
ServerIronADX(config)# vlan 100
ServerIronADX(config-vlan-100)# untagged ethernet 1/1
ServerIronADX(config-vlan-100)# router-interface ve 1
ServerIronADX(config-vlan-100)# exit
ServerIronADX(config)# vlan 200
ServerIronADX(config-vlan-200)# untagged ethernet 1/2
ServerIronADX(config-vlan-200)# router-interface ve 2
ServerIronADX(config-vlan-200)# exit

ServerIronADX-A(config)# interface ve 1
ServerIronADX-A(config-ve-1)# 10.10.20.2 255.255.255.0
ServerIronADX-A(config-ve-1)# ip nat outside
ServerIronADX-A(config-ve-1)# exit
ServerIronADX-A(config)# interface ve 2
ServerIronADX-A(config-ve-2)# 10.10.10.2 255.255.255.0
ServerIronADX-A(config-ve-2)# ip nat inside
ServerIronADX-A(config-ve-2)# exit

ServerIronADX-B(config)# interface ve 1
ServerIronADX-B(config-ve-1)# 10.10.20.4 255.255.255.0
ServerIronADX-B(config-ve-1)# ip nat outside
ServerIronADX-B(config-ve-1)# exit
ServerIronADX-B(config)# interface ve 2
ServerIronADX-B(config-ve-2)# 10.10.10.4 255.255.255.0
ServerIronADX-B(config-ve-2)# ip nat inside
ServerIronADX-B(config-ve-2)# exit
```

4. Configure the static NAT entries on each ServerIron ADX, and assign device ownerships to the NAT entries. In this example, ServerIron ADX A is assigned as the NAT owner, and therefore takes the higher priority value:

```
SI_A(config)# ip nat inside source static 10.10.1.100 10.20.1.100 2
SI_A(config)# ip nat inside source static 10.10.1.101 10.20.1.101 2
SI_A(config)# ip nat inside source static 10.10.1.102 10.20.1.102 2
SI_A(config)# ip nat inside source static 10.10.1.103 10.20.1.103 2

SI_B(config)# ip nat inside source static 10.10.1.100 10.20.1.100 1
SI_B(config)# ip nat inside source static 10.10.1.101 10.20.1.101 1
SI_B(config)# ip nat inside source static 10.10.1.102 10.20.1.102 1
SI_B(config)# ip nat inside source static 10.10.1.103 10.20.1.103 1
```

5. Configure the VRRP-E parameters. The IP address configured with the **ip-address** command is the address that will be backed up by VRRP-E. The **track-port** commands enable tracking on the interfaces on the other side of the ServerIron ADX that complete the link for the VRID. For example, traffic that is addressed to VRID 1 enters the ServerIron ADX through VE interface 1 and leaves the ServerIron ADX through VE interface 2. Under normal circumstances, if VE interface 2 goes down, VRID 1 remains active. When you track interfaces for a VRID, if the state of one of the tracked interfaces changes, the ServerIron ADX associates the change with the VRID interface. For example, if virtual routing interface 2 goes down, the ServerIron ADX

associates this state change with VRID 1 and causes VRRP-E to fail over the VRID to the other ServerIron ADX. The ServerIron ADX on which you configure the higher VRRP-E backup priority becomes the default master for the VRID, while other ServerIron ADX becomes the backup. In this example, ServerIron ADX A is configured as the default master for the HA setup.

```
ServerIronADX-A(config)# interface ve 1
ServerIronADX-A(config-ve-1)# ip vrrp-extended vrid 1
ServerIronADX-A(config-ve-1-vrid-1)# backup priority 200 track-priority 10
ServerIronADX-A(config-ve-1-vrid-1)# ip-address 10.10.20.3
ServerIronADX-A(config-ve-1-vrid-1)# enable
ServerIronADX-A(config-ve-1-vrid-1)# track-port ethernet 1/2
ServerIronADX-A(config-ve-1-vrid-1)# track-port ve 2
ServerIronADX-A(config-ve-1-vrid-1)# exit
ServerIronADX-A(config-ve-1)# exit
ServerIronADX-A(config)# interface ve 2
ServerIronADX-A(config-ve-2)# ip vrrp-extended vrid 2
ServerIronADX-A(config-ve-2-vrid-2)# backup priority 200 track-priority 10
ServerIronADX-A(config-ve-2-vrid-2)# ip-address 10.10.10.3
ServerIronADX-A(config-ve-2-vrid-2)# enable
ServerIronADX-A(config-ve-2-vrid-2)# track-port ethernet 1/1
ServerIronADX-A(config-ve-2-vrid-2)# track-port ve 1
ServerIronADX-A(config-ve-2-vrid-2)# exit
ServerIronADX-A(config-ve-2)# exit

ServerIronADX-B(config)# interface ve 1
ServerIronADX-B(config-ve-1)# ip vrrp-extended vrid 1
ServerIronADX-B(config-ve-1-vrid-1)# backup priority 100 track-priority 10
ServerIronADX-B(config-ve-1-vrid-1)# ip-address 10.10.20.3
ServerIronADX-B(config-ve-1-vrid-1)# enable
ServerIronADX-B(config-ve-1-vrid-1)# track-port ethernet 1/2
ServerIronADX-B(config-ve-1-vrid-1)# track-port ve 2
ServerIronADX-B(config-ve-1-vrid-1)# exit
ServerIronADX-B(config-ve-1)# exit
ServerIronADX-B(config)# interface ve 2
ServerIronADX-B(config-ve-2)# ip vrrp-extended vrid 2
ServerIronADX-B(config-ve-2-vrid-2)# backup priority 100 track-priority 10
ServerIronADX-B(config-ve-2-vrid-2)# ip-address 10.10.10.3
ServerIronADX-B(config-ve-2-vrid-2)# enable
ServerIronADX-B(config-ve-2-vrid-2)# track-port ethernet 1/1
ServerIronADX-B(config-ve-2-vrid-2)# track-port ve 1
ServerIronADX-B(config-ve-2-vrid-2)# exit
ServerIronADX-B(config-ve-2)# exit
```

6. Configure a VIP group on each device and use the **static-nat-ip** command to add the static NAT entry under the VIP group.

```
ServerIronADX(config)# server vip-group 1
ServerIronADX(config-vip-group-[1])# static-nat-ip 10.20.1.100
ServerIronADX(config-vip-group-[1])# static-nat-ip 10.20.1.101
ServerIronADX(config-vip-group-[1])# static-nat-ip 10.20.1.102
ServerIronADX(config-vip-group-[1])# static-nat-ip 10.20.1.103
```

7. Once the dynamic NAT pools are defined as VIP group members, the VIP group must be added to a VRID. It must be added to the outside interface (the interface having ip nat outside configured):

```
ServerIronADX(config)# interface ve1
ServerIronADX(config-ve-1)# ip vrrp vrid 1
ServerIronADX(config-ve-1-vrid-1)# vip-group 1
```

Note that each VIP group can have only one VRID associated with it. Also, each virtual IP address can belong to only one VIP group.

IP NAT session synchronization in HA configurations

IP NAT sessions created by the active ServerIron ADX in a HA configuration are synchronized to the standby ServerIron ADX. When failover occurs, the standby ServerIron ADX will be able to use the IP NAT session information created by the active ServerIron ADX. IP NAT session synchronization is performed automatically. No configuration is necessary.

Displaying NAT information

The following sections describe how to display NAT-related information.

Displaying NAT statistics

To display NAT statistics, enter the **show ip nat statistics** command such as in the following example.

```
ServerIronADX#rconsole 1 1
ServerIronADX1/1#show ip nat statistics
Debug counters:
TCP FWD:
send nat unreachable(tcp fwd) =0  nat tcp no ports avl = 0
nat tcp status zero = 0  nat tcp ip status zero = 0
nat tcp usr index null = 0
 TCP REV:
send nat unreachable(tcp rev) =0  nat tcp rev no ports avl = 0
nat tcp rev status zero = 0  nat tcp rev ip status zero = 0
nat tcp rev usr index null = 0

UDP FWD:
send nat unreachable(udp_fwd) =0  nat udp fwd no ports avl = 0
nat udp fwd status zero = 0  nat udp fwd ip status zero = 0
nat udp fwd usr index null = 0
 UDP REV:
send nat unreachable(udp rev) =0  nat udp rev no ports avl = 0
nat udp rev status zero = 0  nat udp rev ip status zero = 0
nat udp rev usr index null = 0
nat corruption = 0  rtsp port unavailable = 0
RTSP inside alloc same = 0  RTSP reply port not same= 0
Wrong port range = 0

Total translations: 97 (0 static, 97 dynamic)
Hits: 1694 Misses: 78
Expired translations: 1675
Dynamic mappings:
 pool p1: prefix_len= 24
     start 10.1.1.10 end 10.1.1.15
     total addresses 6 overloaded
```

```
[0]: h: 0 t: 0 m: 26e19000 T: 384 f: 384
[1]: h: 0 t: 0 m: 26e2a000 T: 384 f: 384
[2]: h: 0 t: 0 m: 26e2f000 T: 384 f: 384
[3]: h: 0 t: 0 m: 26e31000 T: 384 f: 384
[4]: h: 0 t: 0 m: 26fd0000 T: 384 f: 384
[5]: h: 0 t: 0 m: 26fd5000 T: 384 f: 384
[0]: h: 0 t: 0 m: 26e27000 T: 320 f: 320
[1]: h: 0 t: 0 m: 26e2c000 T: 320 f: 320
[2]: h: 0 t: 0 m: 26fca000 T: 320 f: 320
[3]: h: 0 t: 0 m: 26fcd000 T: 320 f: 320
[4]: h: 0 t: 0 m: 26fd2000 T: 320 f: 320
[5]: h: 0 t: 0 m: 26fd7000 T: 320 f: 320

 [0]: h: 322 t: 314 m: 26f94000 T: 7168 f: 7160
 [1]: h: 373 t: 340 m: 26faf000 T: 7168 f: 7135
 [2]: h: 136 t: 127 m: 27758000 T: 7168 f: 7159
 [3]: h: 240 t: 235 m: 27773000 T: 7168 f: 7163
 [4]: h: 443 t: 410 m: 2778e000 T: 7168 f: 7135
 [5]: h: 157 t: 148 m: 277a9000 T: 7168 f: 7159

Sess: Total 2000000, Avail 1999483, NAT 258
NAT internal session: 258
NAT extended internal session: 0
NAT extended prime internal session: 0
Stream media=1, RTSP=(1:0), MMS=(1:0), PNM=(1:0)

Inside global     Last Inside Local  xmit pkts  xmit bytes rx pkts    rx bytes
cnt
10.1.1.10       4.1.1.72        967      118234    1288      135562    8
10.1.1.11         4.1.1.73          1119       136891     1492       157033
33
10.1.1.12       4.1.1.74         408       49912     544       57256     9
10.1.1.13       4.1.1.75         723       88447     960       101040    5
10.1.1.14         4.1.1.70           1329       162581     1772       186503
```

Syntax: show ip nat statistics

TABLE 15 Display fields for show ip nat statistics

Field	Description
send nat unreachable (tcp fwd)	Indicates the number of times that a “port unreachable” message was generated for NAT TCP forward traffic.
nat tcp no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not get a port from the port pool for a NAT IP for TCP forward traffic.
nat tcp status zero	Indicates the number of times that an error in NAT translation for TCP forward traffic has occurred.
nat tcp ip status zero	Indicates the number of time that an error in NAT translation for TCP forward traffic has occurred.
nat tcp usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not create a user session for TCP forward traffic.
send nat unreachable (tcp rev)	Indicates the number of times that a “port unreachable” message was generated for reverse traffic.

TABLE 15 Display fields for show ip nat statistics (Continued)

Field	Description
nat tcp rev no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not get a port from the port pool for an IP NAT for TCP reverse traffic.
nat tcp rev status zero	Indicates the number of times that an error in NAT translation for TCP reverse traffic has occurred.
nat tcp rev ip status zero	Indicates the number of times that an error in NAT translation for TCP reverse traffic has occurred.
nat tcp rev usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not create a user session for TCP reverse traffic.
send nat unreachable (udp fwd)	Indicates the number of times that a “port unreachable” message was generated for UDP forward traffic.
nat udp fwd no ports avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not get a port from the port pool for a NAT IP for UDP forward traffic.
nat udp fwd status zero	Indicates the number of times that an error in NAT translation for UDP forward traffic has occurred.
nat udp fwd ip status zero	Indicates the number of times that an error in NAT translation for UDP forward traffic has occurred.
nat udp fwd usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not create a user session for UDP forward traffic.
send nat unreachable (udp rev)	Indicates the number of times that a “port unreachable” message was generated for UDP reverse traffic because the ServerIron ADX could not get a port from the port pool for a NAT IP.
nat udp rev no port avl	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not get a port from the port pool for a NAT IP for UDP reverse traffic.
nat udp rev status zero	Indicates the number of times that an error in NAT translation for UDP reverse traffic has occurred.
nat udp rev ip status zero	Indicates the number of times that an error in NAT translation for UDP reverse traffic has occurred.
nat udp rev usr index null	Indicates the number of times that a “port unreachable” message was generated because the ServerIron ADX could not create a a user session for UDP reverse traffic.
sw l4 nat corruption	Indicates the number of instances of NAT session corruption.
rstp port unavailable	Indicates the number of times that a NAT port was not available for RSTP.
RTSP inside alloc same	Indicates the number of times that the used port and proposed client port were the same for RSTP.
RTSP reply port not same	Indicates the number of times that the used port and proposed client port were not the same for RTSP.
Wrong port range	Indicates the number of times that the NAT used a port from the wrong port range. For example, where NAT uses a port from the normal port pool range for RTSP.
Port Pool Parameters	

TABLE 15 Display fields for show ip nat statistics (Continued)

Field	Description
[x]	The variable represented by "x" represents the index of the IP address in the IP NAT pool. For example, [0] refers to the first IP address in the IP pool (10.1.1.10). [1] refers to the second IP address in this IP pool (10.1.1.11).
h	The value following "h:" refers to the head of the port pool for the IP address in the IP NAT pool. The head indicates the location in the port pool where the next port will be allocated from.
t	The value following "t:" refers to the tail of the port pool for the IP address in the IP NAT pool. The tail indicates the location in the port pool where the next port will be freed from.
T	The value following "T:" refers to the total number of ports in the port pool for that IP address in the IP NAT pool.
f	The value following "f:" refers to the number of free ports in the port pool for this IP address.

NOTE
Three ranges are displayed in the output example. The first range from [0] to [5] refers to the MMS port pools for NAT IPs. The second range from [0] to [5] refers to the RTSP port pools for NAT IPs. The third range from [0] to [5] refers to the normal port pools for NAt IPs.

Displaying NAT translation

To display the currently active NAT translations, enter the **show ip nat translation** command.

```
ServerIronADX1/1#sh ip nat translation

Pro Inside global       Inside local        Outside local       Outside global
tcp 10.1.1.14:10861     4.1.1.36:25752      5.1.1.32:80         5.1.1.32:80
tcp 10.1.1.13:10716     4.1.1.48:35175      5.1.1.43:80         5.1.1.43:80
```

Syntax: show ip nat translation

NOTE
You can enter the **show ip nat translation** command only when you use the rconsole to log into a barrel processor (BP). The command is not supported on the main processor CPU.

TABLE 16 Display fields for show ip nat translation

Field	Description
Pro	When PAT is enabled, this field indicates the protocol NAT is using to uniquely identify the host. NAT can map the same IP address to multiple hosts and use the protocol port to distinguish among the hosts. This field can have one of the following values: • **tcp**: When NAT is associating a TCP port with the host on the private network. • **udp**: When NAT is associating a UDP port with the host on the private network.
Inside global	The Internet address mapped to the private address listed in the Inside local field for inside NAT.

TABLE 16 Display fields for show ip nat translation (Continued)

Field	Description
Inside local	The private address mapped to the Internet address listed in the Inside global field for inside NAT.
Outside global	The destination of the traffic. If PAT is enabled, the TCP or UDP port also is shown. **NOTE:** Currently, outside NAT is not supported.
Outside local	The destination of the traffic. If PAT is enabled, the TCP or UDP port also is shown. **NOTE:** Currently, outside NAT is not supported.

Displaying NAT redundancy information

You can display information about the state of the static NAT IP or NAT pool (dynamic), the MAC address used, and the configured priority. The MAC address used for the NAT IP is a special construct, where the last three bytes of the MAC address are derived from the shared NAT IP address (similar to the symmetric MAC).

To display NAT redundancy information, enter the following command.

```
ServerIronADX# show ip nat redundancy (on active)
NAT Pool Start IP: 10.1.1.150 Mac address: 020c.db01.0196
State: Active Priority: High
NAT Pool Start IP: 10.1.1.91   Mac address: 020c.db01.015b
State: Active Priority: High
NAT Pool Start IP: 10.1.1.92   Mac address: 020c.db01.015c
State: Active Priority: High
NAT Pool Start IP: 10.1.1.95   Mac address: 020c.db01.015f
State: Active Priority: High
```

The two **Priority** options are **High** and **Low**. That is, 2 or 1.

The two **State** options are **Active** and **Standby**.

```
ServerIronADX# show ip nat redundancy (on standby)
NAT Pool Start IP: 10.1.1.150 Mac address: 020c.db01.0196
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.91   Mac address: 020c.db01.015b
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.92   Mac address: 020c.db01.015c
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
NAT Pool Start IP: 10.1.1.95   Mac address: 020c.db01.015f
State: Standby Priority: Low
Standby Idle count: 0 Threshold: 20
```

Syntax: show ip nat redundancy

Displaying VRRPE information

To display VRRPE information, enter the following command.

```
ServerIronADX_Lower# show ip vrrp-e brief

Total number of VRRP-Extended routers defined: 2
Interface VRID CurPri P State  Master addr   Backup addr       VIP

 v5         1    125 P Master Local         Unknown         5.1.1.9
 v10        2    125 P Master Local         Unknown         10.1.1.9
```

Syntax: show ip vrrp-e brief

Clearing NAT entries from the table

Use the **clear ip nat** command to manually clear entries from the NAT table.

Syntax: clear ip nat <*protocol*> <*global-ip*> <*global-port*> <*local-ip*> <*local-port*>

The <*protocol*> parameter specifies the protocol type and can be **tcp** or **udp** plus its global or local port number.

To clear a specific NAT entry based on the private and global IP addresses, enter the command such as the following.

```
ServerIronADX# clear ip nat inside 209.157.1.43 10.10.10.5
```

This command clears the inside NAT entry that maps private address 10.10.10.5 to Internet address 209.157.1.43.

Syntax: clear ip nat inside <*global-ip*> <*private-ip*>

To clear all static and dynamic entries from the NAT translation table, enter the following command.

```
ServerIronADX# clear ip nat all
```

Syntax: clear ip nat all